①

~~78 01 50~~

HEURISTIC PROGRAMMING PROJECT

Computer Science Department

Stanford University

Final Technical Report for Contract Period
August 1, 1973 through July 31, 1977

Contract DAHC-73-C-0435

Principal Investigators:
Edward A. Feigenbaum
Joshua Lederberg
Bruce Buchanan

Submitted to Advanced Research Projects Agency
October, 1977

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A131700	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Final Technical Report for Contract Period August 1, 1973 through July 31, 1977		5. TYPE OF REPORT & PERIOD COVERED Technical
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Edward A. Feigenbaum, Joshua Lederberg, Bruce Buchanan		8. CONTRACT OR GRANT NUMBER(s) DAHC15-73-C-0435
9. PERFORMING ORGANIZATION NAME AND ADDRESS Stanford University, Stanford, California 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency, Information Processing Techniques Office, 1400 Wilson Avenue, Arlington, VA. 22209		12. REPORT DATE October 1977
		13. NUMBER OF PAGES 65
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Mr. Philip Surra, Resident Representative, Office of Naval Research, Durand 165, Stanford University		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Reproduction in whole or in part is permitted for any purpose of the U.S. Government

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

artificial intelligence, knowledge-based systems, production systems, knowledge representation, knowledge acquisition, knowledge utilization, learning systems, meta-level knowledge

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The research activities of the Heuristic Programming Project, for the four-year period ending July 31, 1977, are summarized in this report. Contributions to Knowledge Engineering research in the fields of knowledge acquisition (both interactive and sutomated), knowledge representation and knowledge utilization were reported in over thirty publications by members of the project. A summary of those publications is presented here.

The AI Handbook, an encyclopedic reference to the field of Artificial Intelligence

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) is described in the appendix, along with the expected table of contents and sample articles.

DAHC15-73-C-0435 Final Report

HEURISTIC PROGRAMMING PROJECT

Computer Science Department

Stanford University

Final Technical Report for Contract Period

August 1, 1973 through July 31, 1977

Principal Investigators:

Edward A. Feigenbaum
Joshua Lederberg
Bruce Buchanan

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2494
Contract No. DAHC15-73-C-0435

October, 1977

Table of Contents

ABSTRACT

The research activities of the Heuristic Programming Project, for the four-year period ending July 31, 1977, are summarized in this report. Contributions to Knowledge Engineering research in the fields of knowledge acquisition (both interactive and automated), knowledge representation and knowledge utilization were reported in over thirty publications by members of the project. A summary of those publications is presented here.

The AI Handbook, an encyclopedic reference to the field of Artificial Intelligence, is described in the appendix, along with the expected table of contents and sample articles.

KEY WORDS:

Artificial intelligence, knowledge-based systems, production systems, knowledge representation, knowledge acquisition, knowledge utilization, learning systems, meta-level knowledge.

1 Introduction

1.1 Organization of this report

Our completed research results have been reported in many publications which have appeared in technical journals, conference proceedings and internal Heuristic Programming Project (HPP) reports (including Ph.D. theses). Many of these publications report on research which spans the categories of acquisition, representation and utilization of knowledge which comprise our knowledge engineering activities. Consequently organizing a summary of our research along these dimensions would result in considerable redundancy. Moreover, some of our research led into new, yet relevant, areas which were not foreseen when we laid out our specific objectives in the 1973 and 1975 proposals.

An alternative organization is one which is built around the HPP research reports themselves. This organization offers the reader a glimpse of what is in those reports, and informs him where specific details may be found. We have chosen this publication-based approach here; however, we also present, in Sect. 2, a summary of results, organized according to the goals set forth in our 1975 proposal, with pointers to the HPP reports which are summarized in Sect. 3.

A bibliography of HPP reports is given in Sect. 4. The appendix contains a progress report on the AI Handbook, an encyclopedic reference to the subject matter of Artificial Intelligence research. The expected table of contents is given, along with a few sample articles.

1.2 HPP research highlights, 1973 - 1977

Here, in brief, are the major accomplishments of the Heuristic Programming Project during the past four years. Although each of the items in the following list is discussed in more detail in section 3, pointers to the HPP bibliography are given here also. Although the references cited here are all dated from 1975 to 1977, the research itself extends over the full four-year period.

1. An interactive knowledge acquisition program, called TEIRESIAS, which accepts, analyzes and critiques new rules to be incorporated in the MYCIN system, was completed (Davis, 76-7).

2. The concept of meta-rules was developed and implemented

in the MYCIN and TEIRESIAS systems (Davis, 77-6), and generalized to other knowledge-based systems using production rules (Davis, 77-16; Nii, 77-7).

3. The TEIRESIAS program demonstrated, inter alia, the power of using a program's knowledge of its own representations to acquire additional knowledge (Davis, 77-8).

4. The Meta-DENDRAL program for automatic formation of new theoretical rules in mass spectrometry was successfully applied to a real problem in chemistry (Buchanan, 76-4).

5. The techniques for automatic rule formation developed in the Meta-DENDRAL program were extended to a new task domain (Mitchell, 77-4). This extension also involved a generalization of the set of descriptive terms in which the rules can be expressed.

6. Generalized techniques for automatically acquiring new knowledge in the form of production rules were developed (Buchanan, 77-6; Mitchell, 77-13; Smith, 77-14).

7. Studies were made and reported of the processes and representations which facilitate the multiple uses of knowledge in the context of signal understanding (Nii, 77-7).

8. Design alternatives for systems which integrate multiple sources of knowledge were investigated and reported for an application in the domain of protein structure inference (Feigenbaum, 77-15; Engelmore, 77-2).

9. In order to provide a program with the capability of explaining its own reasoning steps, the history-list schema was developed and implemented within the MYCIN system (Shortliffe, 75-2; Davis, 76-7; Clancey, 77-1).

10. A comprehensive survey of the many activities which comprise the field of Artificial Intelligence was initiated and nearly completed during this period. Examples of some of the articles written for this "A.I. Handbook" are given in the appendices.

11. The scope and limitations of building knowledge-based expert systems like MYCIN were studied (Shortliffe, 75-16). The scope of such systems has been demonstrated in a practical sense through its recent application in several new task domains (Feigenbaum, 77-25).

12. A model for incorporating inexact reasoning within knowledge-based systems was developed and used in the MYCIN system (Shortliffe, 75-1).

13. Production system techniques were carefully studied, and a comprehensive survey of the current state of the art in this area was documented (Davis, 75-6, 75-7).

14. A study was made of computer networking as a means of technology transfer (Carhart, 75-14).

15. An investigation of the use of AI techniques to aid in the process of scientific discovery led to a novel system for discovering mathematical relations (Lenat, 76-8).

16. A new concept, called the contract net, was proposed for the design of a highly parallel and highly asynchronous multiprocessor computer organization for AI problem solving (Smith, 77-12; Wiederhold, 77-21).

17. A study was made of distributed data bases (Garcia-Molina, 77-27).

18. A study was made of the use of production systems for automatic deduction (Nilsson, 77-28).

19. A study was made of systems for automatic debugging of user programs, and a prototype system was implemented for debugging simple LISP programs. (Brown, Ph.D. dissertation)

20. Several papers were presented at conferences to transfer the paradigms of knowledge-based expert systems to the AI and scientific communities (Feigenbaum, 74-4, 77-15, 77-25; Buchanan, 75-13; Martin, 77-5).

2 Outline of HPP Research Goals and Relevant Work

2.1 Overview of AI Research

At a 1974 conference of the Federation of Experimental Societies for Experimental Biology, Feigenbaum chaired a session on computer applications. He introduced the session with a definition of Artificial Intelligence, an overview of AI research, examples of high performance programs and a discussion of knowledge-based systems. Because this succinct survey is not easily accessible to th omputer science community, it is reproduced here nearly in its entire .

Artificial Intelligence research is that part o omputer science that is concerned with the symbol-manipulation proc es that produce intelligent action. By "intelligent action" is meant ct or decision that is goal-oriented, arrived at by an understand chain of symbolic analysis and reasoning steps, in which knowledge of the work informs and guides the reasoning.

The potential uses of computers by people to accomplish tasks can be "one-dimensionalized" into a spectrum representing the nature of instruction that must be given the computer to do its job. Call it the "WHAT-TO-HOW" spectrum. At one extreme of the spectrum, the user supplies the intelligence to instruct the machine with precision exactly HOW to do his job step-by-step. Progress in computer science can be seen as steps away from the extreme HOW point on the spectrum: the familiar panoply of assembly languages, subroutine libraries, compilers, extensible languages, etc. At the other extreme of the spectrum is the user with his real problem (WHAT he wishes the computer, as his instrument, to do for him). He aspires to communicate WHAT he wants done in a language that is comfortable to him (perhaps English); via communication nodes that are convenient for him (including, perhaps, speech or pictures); with some generality, some abstractness, perhaps some vagueness, imprecision, even error; without having to lay out in detail all necessary subgoals for adequate performance with reasonable assurance that he is addressing an intelligent agent that is using knowledge of his world to understand his intent, to fill in his vagueness, to make specific his abstractions, to correct his errors, to discover appropriate subgoals, and ultimately to translate WHAT he really wants done into processing steps that define HOW it shall be done by a real computer. The research activity aimed at creating computer programs that act as "intelligent agents" near the WHAT end of the WHAT-TO-HOW spectrum can be viewed as the long-range goal of AI research. Historically, AI research has always been the primary vehicle for progress toward this end, even though science as a whole is largely unaware of the role, the goals, and the progress made.

The root concepts of AI as a science are 1) the conception of the digital computer as a symbol-processing device (rather than as merely a number calculator); and 2) the conception that all intelligent activity can be precisely described as symbol manipulation. (The latter is the fundamental working hypothesis of the AI field, but is controversial outside of the field.) The first inference to be made therefrom is that the symbol manipulations which constitute intelligent activity can be modeled in the medium of the symbol-processing capabilities of the digital computer.

This intellectual advance--which gives realization in a physical system, the digital computer, to the complex symbolic processes of intelligent action and decision, with detailed case studies of how the realization can be accomplished, and with bodies of methods and techniques for creating new demonstrations--ranks as one of the great intellectual achievements of science, allowing us finally to understand how a physical system can also embody mind. The fact that large segments of the intellectual community do not yet understand that this advance has been made does not change its truth or its fundamental nature.

2.1.1 HIGH PERFORMANCE PROGRAMS THAT PERFORM AT NEAR-HUMAN LEVEL IN SPECIALIZED AREAS

As the AI research matured to the point where the practitioners felt comfortable with their tools, and adventuresome in their use; as the need to explore the varieties of problems posed by the real world was more keenly felt, and as the concern with knowledge-driven programs (see below) intensified, specific projects arose which aimed at and achieved levels of problem solving performance that equaled, and in some cases exceeded, the best human performance in the tasks being studied. An example of such a program is the heuristic DENDRAL program, developed by our interdisciplinary group at Stanford University. It solves the scientific induction problem of analyzing the mass spectrum of an organic molecule to produce a hypothesis about the molecule's total structure. This is a serious and difficult problem in a relatively new area of analytical chemistry. The program's performance has been generally very competent and in "world's champion" class for certain specialized families of molecules. Similar levels of successful performance have been achieved by some of the MATHLAB programs at MIT that assist scientists in doing symbolic integration in calculus, and are virtually unexcelled. In medical problem solving tasks, other applications are beginning to be developed. One of these is the MYCIN program (being developed at the Stanford University Medical School) for advising physicians about antibiotic therapy for treating infectious diseases.

2.1.2 KNOWLEDGE-BASED SYSTEMS

This term is intended to connote, in familiar terms, something like: "What is the meaning of..." or "How is that to be understood..." or "What knowledge about the world must be brought to bear to solve the particular problem that has just come up?" The research deals with the problem of extracting the meaning of: utterances in English; spoken versions of these; visual scenes; and other real-world symbolic and signal data. It aims toward the computer understanding of these as evidenced by the computer's subsequent linguistic, decision-making, question-answering, or motor behavior.

Thus for example, we will know that our "intelligent agent" understood the meaning of the English command we spoke to it if: a) the command was in itself ambiguous; b) but was not ambiguous in context; and c) the agent performed under the appropriate interpretation and ignored the interpretation that was irrelevant in context.

One paradigm for work in this area of knowledge-based systems is, very generally sketched, as follows:

a) a situation is to be described or understood; a signal input is to be interpreted; or a decision in a problem-solution path is to be made. Examples: A speech signal is received and the question is, "What was said?" The TV camera system sends a quarter-million bits to the computer and the question is, "What is out there on that table and in what configuration?" The molecule structure-generator must choose a chemical functional group for the "active center" of the molecular structure it is trying to hypothesize and the question is, "What does the mass spectrum indicate is the 'best guess'?"

b) Specialized collections of facts about the various particular task domains, suitably represented in the computer memory (call these Experts) can recognize situations, analyze situations, and make decisions or take actions within the domain of their specialized knowledge. Examples: In the Carnegie-Mellon University Hear-say speech understanding system, currently the Experts that contribute to the current best hypothesis are an acoustic-phonetic Expert, a grammar Expert, and a chess Expert (since chess playing is the semantic domain of discourse). In heuristic DENDRAL, the Experts are those that know about stability of organic molecules in general, mass spectrometer fragmentation processes in particular, nuclear magnetic resonance phenomena, and so on.

For each of the sources of knowledge that can be delineated, schemes must be created for bringing that knowledge to bear at some place in the ongoing analysis or understanding process. The view is held that programs should take advantage of a wide range of knowledge, creating islands of certainty as targets of opportunity arise, and using these as anchors for further uncertainty reduction. It is an expectation that always some different aspect provides the toehold for making headway--that is, that unless a rather large amount of knowledge is available and ready for application, this paradigmatic scheme will not work at all.

To summarize: In AI research, there are foci upon the encoding of knowledge about the world in symbolic expressions so that this knowledge can be manipulated by programs; and the retrieval of these symbolic expressions, as appropriate, in response to demands of various tasks. AI has sometimes been called "applied epistemology" or "knowledge engineering."

The AI field has come increasingly to view as its main line of endeavor: knowledge representation and use, and an exploration of understanding (how symbols inside a computer, which are in themselves essentially abstract and contentless, come to acquire a meaning).

The impact of the "knowledge engineering" research and development endeavors of the AI scientists can and ultimately will have a major impact on the organization of specialized knowledge in all the domains it touches; on intelligent computer assistance to practitioners; and on the transmission of this knowledge to new students of the domain.

2.2 Knowledge Acquisition

2.2.1 Interactive Knowledge Acquisition

Relevant research results:

Development and implementation of the TEIRESIAS program

Relevant publications: 76-7,77-8,77-9

2.2.2 Automatic Knowledge Acquisition: Transfer of Research Results to the Scientific Community

Relevant research results:

New contributions to mass spectrometry and 13C NMR spectroscopy by Meta-DENDRAL program and its generalization;

Development of the version space concept and, from it, a more general approach to automated learning techniques.

Relevant publications: 74-3,76-4,77-4,77-6,77-13,77-14

2.3 Knowledge Representation: Models for Theory Formation

Relevant research results:

Extension of Meta-DENDRAL to new application domain, using different descriptive terms;

Development and Implementation of meta-level knowledge in TEIRESIAS;

Study of scope and limitations of production rule systems.

Relevant publications: 75-6,75-7,76-7,77-7,77-8,77-16

2.4 Knowledge Utilization

2.4.1 Multiple Uses of the Same Knowledge Base

Relevant research results:

Development of an "intelligent teacher" for the MYCIN system (work still in progress);

Development and implementation of schema rules and an explanation subsystem to permit a program's knowledge of its own representations to be exploited for explanation and acquisition of new knowledge;

Development of the SU/X and SU/P programs which generalize the meta-rule concept to other knowledge-based systems.

Relevant publications: 76-7,77-2,77-7,77-8

2.4.2 Program Strategy and Control

Relevant research results:

MYCIN was extended to use meta-rules;

Meta-rules were added and tested in MYCIN;

Meta-rules were used in the construction of the TEIRESIAS program and the concept was generalized to both goal-driven and event-driven production rule systems.

The contract net concept was developed to control problem solving in a network of asynchronous parallel processors.

Relevant publications: 76-7,77-7,77-12,77-16

2.4.3 Integration of Multiple Sources of Knowledge

Relevant research results:

Design alternatives were investigated for using multiple KSs in the protein structure inference system (CRYSALIS);

Implementations of two systems for protein structure inference, one primarily model-driven and the other primarily data-driven, are currently in progress;

Integration of KSs and data bases distributed among many computing systems was investigated, leading to the development of the contract net.

Relevant publications: 76-8,77-2,77-12,77-15,77-27

2.4.4 Explanation

Relevant research results:

The scope of the MYCIN-like explanation scheme was studied;

A general history-list schema was developed and incorporated in TEIRESIAS, including procedures for adding items to the history-list, accessing relevant items, and describing the role of any item in the program's line of reasoning.

Relevant publications: 75-2,76-7,77-1

3 A Summary of HPP research reports, 1973 through 1977

In this section we present summaries of the Heuristic Programming Project activities, as reported in our publications. The report summaries are grouped according to the sub-projects which comprise the total effort, and chronologically within each sub-project category.

3.1 Results of Investigations with the MYCIN and TEIRESIAS programs

3.1.1 HPP-75-1

Title: A Model of Inexact Reasoning in Medicine

Authors: E. H. Shortliffe and B. G. Buchanan

This report presents an approach to reasoning under uncertainty. The algorithms developed for this purpose were incorporated in the MYCIN program. Although the context of the presentation is in the medical domain, the techniques apply to any task domain in which judgmental knowledge is used.

Often a task domain suffers from having so few data and so much imperfect knowledge that a rigorous probabilistic analysis, the ideal standard by which to judge the rationality of a problem solver's effort, is seldom possible. Practitioners nevertheless seem to have developed an ill-defined mechanism for reaching decisions despite a lack of formal knowledge regarding the interrelationships of all the variables that they are considering. This report proposes a quantification scheme which attempts to model the inexact reasoning process of medical experts. The numerical conventions provide what is essentially an approximation to conditional probability, but ofer advantages over Bayesian analysis when they are utilized in a rule-based computer diagnostic system. One such system, a clinical consultation program named MYCIN, is described in the context of the proposed model of inexact reasoning.

3.1.2 HPP-75-2

Title: Computer-Based Consultations in Clinical Therapeutics; Explanation and Rule Acquisition Capabilities of the MYCIN System

Authors: E. H. Shortliffe, R. Davis, S.G. Axline, B.G. Buchanan, C.C. Green, S.N. Cohen

Techniques for giving a knowledge-based system the capability of explaining its line of reasoning were first explored in the context of the MYCIN system by Shortliffe, Davis, et al. The explanation subsystem of MYCIN understands simple English questions and answers them, in English, in order to justify its decisions or instruct the user.

The interactive explanation capability of MYCIN circa 1974 was first reported in this paper, published in Computers and Biomedical Research. The techniques of retrieving the current subgoals in answer to WHY questions, or of examining preconditions in the current rule in answer to HOW questions are clearly of general applicability to knowledge-based consultation systems.

3.1.3 HPP-76-7

Title: Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases

Author: R. Davis

Significant progress was made in the methodology of interactive knowledge acquisition by Randall Davis as part of his doctoral dissertation. Davis designed and built a program, called TEIRESIAS, which establishes an intelligent link between a human expert and a computer consultant.

Knowledge Acquisition

Knowledge acquisition is described as a process of information transfer from the expert to a performance program, in which TEIRESIAS's task was to "listen" as attentively and intelligently as possible. The process was set in the context of a shortcoming in the knowledge base as an aid to both the expert and the system. The context provides the expert with a useful organization and focus. He is not simply asked to describe all he knows about a domain. He is instead faced with a specific consultation whose results he finds incorrect, and has available a set of tools that will allow him to uncover the system's knowledge, and the rationale behind its performance. His task is then to specify the particular difference between the system's knowledge and his own that accounts for the discrepancy in results. The system relies on the context of the error to form a set of expectations about the character of the information that will be forthcoming. This leads to better comprehension of the expert's statement of that information, and provides a number of checks on its content that insure it will in fact repair the problem discovered. In a single phrase, information transfer in the context of a shortcoming in the knowledge base characterizes our approach to the problem, and suggests the source of many of the system's abilities.

cquiring new rules

New rule acquisition is seen in terms of model-directed understanding and a recognition-oriented approach to comprehension. This means that the system has some model of the content of the signal it is trying to interpret, and uses this to help constrain the set of interpretations it will consider. In our case, the model took the form of rule models, constructs which offer a picture of a "typical" rule of a given type. They are assembled by the system itself, using a primitive, statistically-oriented form of concept formation to produce abstract descriptions of regularities in subsets of rules.

As noted, the context provided by the process of tracking down the error in the knowledge base makes it possible for TEIRESIAS to form expectations about the character of the new rule. These expectations are expressed by selecting a specific rule model. The text of the new rule is then allowed to suggest interpretations (the bottom-up, data directed part of the process), but these are constrained and evaluated for likely validity by reference to the rule model (the top down, hypothesis-driven part). It is the intersection of these two information sources, approaching the task from different directions, that is responsible for much of the system's performance.

Further application of the model-directed formalism is seen in the system's ability to "second guess" the expert. Since it has a model of its own knowledge - the rule models - it can tell when something "fits" in its knowledge base. It is the occurrence of a partial match between the new rule and the model chosen earlier that prompts the system to make suggestions to the expert. This idea of an incompletely met expectation is not specific to the current organization or structure of the rule models, and can be generated to cover any aspects of a representation about which expectations can be formed.

Several implications were seen to follow from the fact that the system itself constructs the rule models from current contents of the knowledge base. Since the process is automated, the expert never has to enter a model by hand; he may even be unaware of their existence. Moreover, unlike most other model-based systems, new models are constructed on the basis of past experience, since rules learned previously are used in forming new models. Since the models are updated as each new rule enters the knowledge base, the model set is kept current, evolving with the growing knowledge base, and reflecting the shifting patterns it contains.

Other implications follow from the fact that these models give the system an abstract picture of its own knowledge base. It means that, in a rudimentary way, the system "knows what it knows, and knows where it is ignorant." It can answer questions about the content of its knowledge base by "reading" a rule model, giving a picture of global structure of its knowledge about a topic. Since the models are ordered on the basis of an empirically defined "strength", the system can also give some indications about likely gaps in its knowledge.

Finally, the coupling of model formation with the model-directed understanding process offers a novel form of closed-loop behavior. Existing rule models are used to guide the acquisition process, the new rule is added to the knowledge base, and the relevant rule models are recomputed. Performance of the acquisition routines may thus conceivably be improved on the very next rule.

In summary then, the system constructs models of its own knowledge on the basis of experience, keeps those models up to date with the current knowledge base, and then uses them in the acquisition of new knowledge later on.

Acquiring New Conceptual Primitives

The primary issue here is the representation and use of knowledge about representations. The schemata and associated structures offer a language for expression of the knowledge and a framework for its organization. There are three levels to that organization: (i) the fundamental unit of organization is the individual schema, a record-like structure that provides the basis for assembling a variety of information about a particular representation; (ii) the schema network is a generalization hierarchy that indicates the existing categories of data structures and relationships between them; and (iii) the slotnames and slotexperts that make up a schema deal with specific representation conventions at the programming language level.

Unlike standard records, however, the schemata and all associated structures are a part of the system itself, available for examination and reference. They also have the ability to describe a certain amount of variability in structure description.

The process of acquiring a new conceptual primitive strongly resembles the creation of a new instance of a record, but has been extended in several ways. It has been made interactive to allow the expert to supply information about the domain, the dialog is couched in high level terms to make comprehensible to a non-programmer, and the whole process has been made as easy and "intelligent" as possible, to ease the task of assembling large amounts of knowledge.

This process was applied to MYCIN by viewing each of the representational primitives as an extended data type, and constructing the appropriate schema for each of them. That is, the language for describing representations was used to formalize a range of information about MYCIN's current set of representations.

The generality of this approach results from a stratification and isolation of different varieties of knowledge at different levels: instances of individual schemata form the collection of domain specific knowledge; the schemata themselves define a base of representation specific information; while the schema-schema supplies a small foundation of representation independent knowledge. This stratification makes it possible for the system to acquire both new instances of existing representations (as in learning about a new organism), and new types of representations (as in the acquisition of a new schema), using a single formalism and body of code.

Finally, it was noted that the same motivation was responsible for both the schemata and the recursive application of the idea to produce the schema-schema. The schemata were designed to automate the handling of the large number of details involved in the creation and management of data structures. But they themselves were sufficiently complex and detailed data structures that it was useful to have a similar device for their construction and management. This resulted in the creation of the schema-schema, and it, along with a small body of associated structures, forms a body of representation independent knowledge from which a knowledge base can be constructed.

3.1.4 HPP-77-1

Title: Explanation Capabilities of Knowledge-Based Production Systems

Authors: A. C. Scott, W. Clancey, R. Davis, and E. H. Shortliffe

This paper discusses the general characteristics of explanation systems: what types of explanations they should be able to give, what types of knowledge will be needed in order to give these explanations, and how this knowledge might be organized. The explanation facility in MYCIN is discussed as an illustration of how the various problems might be approached.

The case is made here for exploring the production system architecture to facilitate program-generated explanations. The process of trying rules and taking actions can be thought of as "reasoning", and explanations consist of showing how rules used information provided by the user to make various intermediate deductions and finally to arrive at the answer. If the information contained in these rules is sufficient to show why an action was taken (without getting into programming details), an explanation can consist of printing each rule that was used (or an English equivalent of what the rule means.)

The development of an explanation mechanism for a consultation system is very much related to the problems of representing knowledge and of making use of different sources of knowledge. Since the production system formalism provides a unified way to represent modular pieces of knowledge, the task of designing an explanation capability is simplified for production-based consultation systems. The example of MYCIN shows how this can be done and illustrates further that a system designed for a single domain with a small, technical vocabulary can give comprehensive answers to a wide range of questions without sophisticated natural-language processing.

3.1.5 HPP-77-8

Title: Knowledge Acquisition on Rule-based Systems: Knowledge about Representations as a Basis for System Construction and Maintenance

Author: Randall Davis

One of the knowledge acquisition issues explored in Davis' work on TEIRESIAS was how to acquire new conceptual primitives with which new inference rules can be built. In a paper to appear in the book Pattern-Directed Inference Systems [D. Waterman and R. Hayes-Roth (eds.), Academic Press], Davis shows that by providing a program with a store of knowledge about its own representations, this acquisition of new concepts can be carried out in a high-level dialog that transfers information efficiently. The necessary knowledge about representations includes both structural and organizational information, and is specified in a data structure schema, a device used to describe representations.

3.1.6 HPP-77-9

Title: Interactive Transfer of Expertise I: Acquisition of New Inference Rules

Author: Randall Davis

Those aspects of Davis' thesis specifically related to knowledge acquisition were extracted and presented in this paper delivered at the 5th IJCAI. He shows an example of TEIRESIAS in operation, and explains how the program guides the acquisition of new inference rules. The concept of meta-level knowledge is described, and an illustration is given of its utility and contribution to the creation of intelligent programs. The following is an overview of the main ideas:

1. Knowledge acquisition in context

Performance programs of the sort TEIRESIAS helps create will typically find their greatest utility in domains where there are no unifying laws on which to base algorithmic methods. In such domains an expert specifying a new rule may be codifying a piece of knowledge that has never previously been isolated and expressed as such. Since this is difficult, anything which can be done to ease the task will prove very useful.

In response, we have emphasized knowledge acquisition in the context of a shortcoming in the knowledge base. To illustrate its utility, consider the difference between asking the expert

What should I know about the stock market?

and saying to him

Here is an example in which you claim the performance program made a mistake. Here is all the knowledge the program used, here are all the facts of the case, and here is how it reached its conclusions. Now, <u>what is it that you know and the system doesn't</u> that allows you to avoid making that same mistake?

Note how much more focussed the second question is, and how much easier it is to answer.

2. Building expectations

The focussing provided by the context is also an important aid to TEIRESIAS. In particular, it permits the system to build up a set of expectations concerning the knowledge to be acquired, facilitating knowledge transfer and making possible several useful features illustrated in the trace and described below.

3. Model-based understanding

Model-based understanding suggests that some aspects of understanding can be viewed as a process of matching: the entity to be understood is matched against a collection of prototypes, or models, and the most appropriate model selected. This sets the framework in which further interpretation takes place, as that model can then be used as a guide to further processing.

While this view is not new, TEIRESIAS employs a novel application of it, since the system has a model of the knowledge it is likely to be acquiring from the expert.

4. Giving programs a model of their own knowledge

The combination of TEIRESIAS and the performance program amounts to a system which has a picture of its own knowledge. That is, it not only knows something about a particular domain, but in a primitive sense it knows what it knows, and employs that model of its knowledge in several ways.

5. Learning by experience

One of the long-recognized potential weaknesses of any model-based system is dependence on a fixed set of models, since the scope of the program's "understanding" of the world is constrained by the number and type of models it has. The models TEIRESIAS employs are not hand-crafted and static, but are instead formed and continually revised as a by-product of its experience in interacting with the expert.

3.1.7 HPP-77-16

Title: Meta-Level Knowledge: Overview and Applications

Authors: Randall Davis and Bruce G. Buchanan

This paper explores a number of issues involving representation and use of what we term meta-level knowledge, or knowledge about knowledge. In the most general terms, meta-level knowledge is knowledge about knowledge. Its primary use here is to enable a program to "know what it knows", and to make multiple uses of its knowledge. That is, the program is not only able to use its knowledge directly, but may also be able to examine it, abstract it, reason about it, or direct its application. To see in general terms how this can be accomplished, imagine taking some of the available representation techniques and turning them in on themselves, using them to describe their own encoding and use of knowledge. In very general terms this is what we have done, with both existing representations and a number of newly developed ones.

Four specific types of meta-level knowledge are described, followed by examples of each:

KNOWLEDGE ABOUT ***************	IS ENCODED IN *************
inference rules	rule models
representation of objects	schemata
representation of functions	function templates
reasoning strategies	meta-rules

The examples reviewed illustrate a number of general ideas about knowledge representation and use which have become evident as a result of experience in building large, high performance programs.

We have, first, the notion that knowledge in programs should be made explicit and accessible. Use of production rules to encode the object level knowledge is one example of this, since knowledge in them is more accessible than that embedded in the code of a procedure. The schemata, templates, and meta-rules illustrate he point also, since each of them encodes a form of information that is, typically, either omitted entirely or at best is left implicit. By making knowledge explicit and accessible, we make possible a number of useful abilities. The schemata and templates, for example, support system maintenance and knowledge acquisition. Meta-rules offer a means for explicit representation of the decision criteria used by the system to select its course of action Subsequent "playback" of those criteria can then provide a form of explanation of the motivation for system behavior. That behavior is also more easily modified, since the information on which it is based is both clear (since it is explicit) and retrievable (since it is accessible). Finally, more of the system's knowledge and behavior becomes open to examination, especially by the system itself.

Second, there is the idea that programs should have access to and an "understanding" of their own representations. To put this another way, consider that over the years numerous representation schemes have been proposed and have generated a number of discussions of their respective strengths and weaknesses. Yet in all these discussions, one entity intimately concerned with the outcome has been left uninformed: the program itself. What this suggests is that we ought to describe to the program a range of information about the representations it employs, including such things as their structure, organization, and use.

But suppose we could describe to a system its representations? What benefits would follow? The primary thing this can provide is a way of effecting multiple uses of the same knowledge. Consider for instance the multitude of ways in which the object level rules have been used. They are executed as code in order to drive the consultation; they are viewed as data structures, and dissected and abstracted to form the rule models; they are dissected and examined in order to produce explanations; they are constructed during knowledge acquisition; and finally they are reasoned about by the meta-rules.

It is important to note here that the feasibility of such multiplicity of uses is based less on the notion of production rules per se, than on the availability of a representation with a small grain size and a simple syntax and semantics. "Small", modular chunks of code written in a simple, heavily stylized form (though not necessarily a situation-action form), would have done as well, as would any representation with simple enough internal structure and of manageable size. The introduction of greater complexity in the representation, or the use of a representation that encoded significantly larger "chunks" of knowledge would require more sophisticated techniques for dissecting and manipulating representations than we have developed thus far. But the key limitations are size and complexity of structure, rather than a specific style of knowledge encoding.

Two other benefits may arise from the ability to describe representations. We noted earlier that much of the information necessary to maintain a system is often recorded in informal ways, if at all. If it were in fact convenient to record this information by describing it to the program itself, then we would have an effective and useful repository of information. We might see information that was previously folklore or informal documentation becoming more formalized, and migrating into the system itself. We have illustrated above a few of the advantages this offers in terms of maintaining a large system.

This may in turn produce a new perspective on programs. The initial scarcity of hardware resources instilled in programmers a certain mania for minimizing machine resources consumed, as evidenced, for example, by the belief that numeric expressions should be evaluated by the programmer and his desk calculator, rather than "waste" the computer's time. More recently, this has meant a certain style of programming in which a programmer spends a great deal of time thinking about a problem first, trying to solve as much as possible by hand, and then abstracting out only the very end product of all of that to be embodied in the program. That is, the program becomes simply a way of manipulating symbols to provide "the answer", with little indication left of what the original problem was, or more important, what knowledge was required to solve it.

But what if we reversed this trend, and instead view a program as a place to store many forms of knowledge about the problem and the proposed solution (i.e., the program itself). This would apply equally well to code and data structures, and could help make possible a wider range of useful capabilities of the sort illustrated above.

One final observation. As we noted at the outset, interest in knowledge-based systems was motivated by the belief that no single, domain independent paradigm could produce the desired level of performance. It was suggested instead that a large store of domain specific (object level) knowledge was required. We might similarly suggest that this too will eventually reach its limits, and that simply adding more object level knowledge will no longer, by itself, guarantee increased performance. Instead it may be necessary to focus on building stores of meta-level knowledge, especially in the form of strategies for effective use of knowledge. Such "meta-level knowledge based" systems may represent a profitable direction for the development of high performance programs.

3.1.8 HPP-77-33

Title: Generalized Procedure Calling and Content-Directed Invocation

Author: Randall Davis

In this paper, Davis continues his exploration of meta-level knowledge, specifically the use of meta-rules to encode problem solving strategies. Meta-rules contain knowledge about how to select from among a set of plausibly useful knowledge sources. Experience with their use in the TEIRESIAS program has demonstrated their utility as a flexible programming mechanism for controlling the invocation of object-level knowledge. The advantages of addressing object-level rules by their content rather than by their labels is also discussed.

3.2 Research in Theory Formation

3.2.1 HPP-76-4

Title: Applications of Artificial Intelligence for Chemical Inference XXII. Automatic Rule Formation in Mass Spectrometry by Means of the Meta-DENDRAL Program

Authors: B. G. Buchanan, D. H. Smith, W. C. White, R. J. Gritter, E. A. Feigenbaum, J. Lederberg and Carl Djerassi

This paper was the first official communication of the theory formation program, Meta-DENDRAL, to the community of analytical chemists. The importance of this work is summarized in the conclusion:

We have shown that the Meta-DENDRAL program is capable of rationalizing the mass spectral fragmentations of sets of molecules in terms of substructural features of the molecules. On known test cases, aliphatic amines and estrogenic steroids, the Meta-DENDRAL program rediscovered the well-characterized fragmentation processes reported in the literature. On the three classes of ketoandrostanes for which no general class rules have been reported, the mono-, di-, and triketoandrostanes, the program found general rules describing the mass spectrometric behavior of those classes. The general rules shown in Tables II, IV, and VI explain many of the significant ions for compounds in these classes while predicting few spurious ions. The program has discovered consistent fragmentation behavior in sets of molecules which have not appeared by manual examination to behave homogeneously in the mass spectrometer.

Programs with knowledge of the scientific domain can provide "smart" assistance to working scientists, as shown by the reasoned suggestions this program makes about extensions to mass spectrometry theory. We are aware that the program is not discovering a new framework for mass spectrometry theory; to the contrary, it comes close to capturing in a computer program all we could discern by observing human problem-solving behavior. It is intended to relieve chemists of the need to exercise their personal heuristics over and over again, and thus we believe it can aid chemists in suggesting more novel extensions to existing theory. It can be argued that the two-dimensional connectivity model of molecules used in this study is not the right model for mass spectrometry; that there are deeper rationalizations of a fragmentation process than subgraph environments. However, this model is commonly used by working chemists and once fragmentations based on this model are defined, chemists can readily provide the remaining "mechanistic" rationalizations or see that further experimental work with labeled compounds is necessary.

Recent statistical pattern recognition work addresses some of the points on rule formation and spectrum prediction raised in this paper. We have avoided blind statistical methods for three important reasons. 1) We wish to explore thousands of possible subgraphs with associated features, as we search for those which are in some way important. Current pattern recognition procedures are restricted to much smaller numbers of manually (or computer-assisted) selected features, adding additional bias to the procedure. 2) We want to know how certain rules were obtained by the program and why certain other rules were rejected or not detected. We can trace the reasoning steps of the Meta-DENDRAL program and determine chemically meaningful answers to such questions in a way that is not possible with purely statistical programs. 3) We wish to constrain the rule formation activity in ways that are natural to a working chemist. For example we may want the program to avoid fragmentations involving aromatic rings or two bonds to the same atom, or, as mentioned above, we may want to look at fragmentations accompanied by loss of CO or other neutral fragments.

Rules can be formulated to explain data in terms that are known to be meaningful to chemists; most importantly, the rule formation constraints are under the control of the chemist. Also we feel that this approach provides a high level of generality in describing fragmentation processes. Although the rules are developed in the context of a particular group of compounds, they are not tied to that group but can be applied in other contexts, or compared to rules developed from other groups of compounds in a search for common features of the rules. For these reasons, we believe that the Meta-DENDRAL program offers a powerful and useful complement to pattern recognition programs for finding relationships between structures and spectral data.

3.2.2 HPP-77-4

Title: Applications of Artificial Intelligence for Chemical Inference XXV. A Computer Program For Automated Empirical 13C NMR Rule Formation

Authors: Tom M. Mitchell and Gretchen M. Schwenzer

To demonstrate the generality of the methods developed for automatic rule formation, a new application domain was selected. The new task was the generation of empirical rules for associating the data from ^{13}C NMR experiments with local structural environments in a molecule. A program was written for this purpose, rules were generated for a chosen set of experiments and then applied to data not included in the training set. For two classes of compounds (acyclic amines and paraffins) the program ranked the correct structure either first or second in 33 of the 35 tests.

3.2.3 HPP-77-6
Title: Model-Directed Learning of Production Rules

Authors: Bruce G. Buchanan and Tom M. Mitchell

This paper describes the learning strategy employed in Meta-DENDRAL, and is a predecessor to Mitchell's work on version spaces. The Meta-DENDRAL program is described in general terms, defining the syntax and semantics of the rules employed therein.

In contrast to statistical approaches, Meta-DENDRAL utilizes a semantic model of the domain. This model has been included for two important reasons. First, it provides guidance for the rule formation program in a space of rules that is much too large to search exhaustively, especially when the input data have ambiguous interpretations. Second, it provides a check on the meaningfulness of the associations produced by the program, in a domain where the trivial or meaningless associations far outnumber the important ones.

The learning program is based on a generator of production rules of a predetermined syntax operating under the constraints of a semantic world model. In common with other induction programs, it also contains an instance selection component and a critic for evaluating potential rules.

The learning cycle is a series of "plan-generate-test" steps as found in many AI systems. After pre-scanning a set of several hundred I/O pairs, the program searches the space of rules for plausible explanations and then modifies the rules on the basis of detailed testing. When rules generated from one training set are added to the model, and a second (or next) block of data examined, the rule set is further extended and modified to explain the new data. That is, the program can now iteratively modify rules formed from the initial training set (and add to them), but it is currently unable to "undo" rules.

3.2.4 HPP-77-13
Title: Version Spaces: A Candidate Elimination Approach to Rule Learning

Author: Tom M. Mitchell

Meta-DENDRAL continues to play an important role as a test bed for exploring new ideas in automatic theory formation. Mitchell has developed a new approach to rule learning which is guaranteed to find, without backtracking, all rule versions consistent with a set of positive and negative training instances. The algorithm put forth uses a representation of the space of those rules consistent with the observed training data. This "rule version space" is modified in response to new training instances by eliminating candidate rule versions found to conflict with each new instance. The use of version spaces is discussed in the context of Meta-DENDRAL.

The program describes the method of representing version spaces in terms of maximally general and maximally specific rule sets. This representation appears to be well suited for learning rules from sequentially presented training instances. A candidate elimination algorithm has been shown which will find all rule versions consistent with all training instances. Backtracking is not required for noise-free training instances, and the final result is independent of the order of presentation of instances.

Version spaces provide at once a compact summary of past training instances and a representation of all plausible rule versions. Because they provide an explicit representation for the space of plausible rules, version spaces allow a program to represent "how much it doesn't know" about the correct form of the rule. This suggests the utility of the version space approach to problems such as intelligent selection of training instances and merging sets of independently generated rules.

3.2.5 HPP-77-14
Title: A Model For Learning Systems

Authors: Reid G. Smith, Tom M. Mitchell, Richard Chestek, and Bruce G. Buchanan

Further study of learning systems (LSs) has led to the development of a general model for the design of such systems. The model provides a common vocabulary for describing different types of learning systems which operate in a variety of task domains. It encourages classification and comparison of LSs and helps identify unique or strong features of individual systems. We believe the model is a useful conceptual guide for LS design, because it isolates the essential functional components, and the information that must be available to these components. The model also suggests a layered architecture for learning at different levels of abstraction.

The learning system has been used to characterize several existing systems: Meta-DENDRAL, Winston's program to learn to identify block structures, Samuel's checker playing program, Waterman's program for learning poker strategy, and London's adaptive control system.

3.3 Studies of Production Rule Systems

3.3.1 HPP-75-7

Title: An Overview of Production Systems

Authors: Randall Davis and Jonathan King

One of the most widely used programming tools in knowledge engineering is the production system. In 1975, Davis and King surveyed the state of the art in PS methodology. Their overview defined two distinct classes of PS's: those used in psychological modelling research (e.g., PSG) and those used in high performance knowledge-based systems (e.g., MYCIN). Although the two technologies are similar, the goals are quite different for the two applications.

The overview continues with some speculations on the nature of appropriate and inappropriate domains for PSs, characterizes the common elements of all PSs, and presents a taxonomy for PSs, selecting four dimensions of characterization.

For those wishing to build knowledge-based expert systems, the homogeneous encoding of knowledge offers the possibility of automating parts of the task of dealing with the growing complexity of such systems. Knowledge in production rules is both accessible and relatively easy to modify. It can be executed by one part of the system as procedural code, and examined by another part as if it were a declarative expression. Despite the difficulties of programming PSs, and their occasionally restrictive syntax, the fundamental methodology at times suggests a convenient and appropriate framework for the task of structuring and specifying large amounts of knowledge. It may thus prove to be of great utility in dealing with the problems of complexity encountered in the construction of large knowledge bases.

3.3.2 HPP-75-6

Title: Production Rules as a Representation of a Knowldge-Based Consultation Program

Authors: Randall Davis, Bruce B. Buchanan and Edward Shortliffe

The appropriateness of the production rule methodology for a knowledge-based consultation program is explored in this paper, generalizing from the MYCIN example. After defining their three design goals of usefulness (including competence), maintenance of an evolving knowledge base, and support of an interactive consultation, the authors report:

Our experience has suggested that production rules offer a knowledge representation that greatly facilitates the accomplishment of these goals. Such rules are straightforward enough to make feasible many interesting features beyond performance, yet powerful enough to supply significant problem solving capabilities. Among these features are the ability for explanation of system performance, and acquisition of new rules, as well as the general 'understanding' by the system of its own knowledge base.

3.3.3 <u>HPP-77-28</u>

Title: A Production System for Automatic Deduction

Author: Nils Nilsson

Logical deduction is a basic activity in many artificial intelligence (AI) systems. Specific applications in which deduction plays a major role include question-answering, program verification, mathematical theorem proving, and reasoning about both mundane and esoteric domains.

In this paper, Nilsson proposes a deduction system that enjoys most of the power of the formal logical systems without embracing their inefficient uniformity. It uses specialized, domain-dependent inference rules that are encoded as <u>productions</u>. As with most production systems, it can easily be modified and extended by adding new production rules or by modifying old ones. The system is based on a synthesis of several ideas from various authors in artificial intelligence and automatic theorem proving.

The advantages of production systems (Davis and King, HPP-75-7) are sufficiently impressive that one would like to model the design on that paradigm. Previous production system designs for deduction systems, however, had somewhat limited logical power. (An example is the restriction to Horn clauses in Kowalski, <u>Logic for Problem Solving</u>, University of Edinburgh School of Artificial Intelligence Memo No. 75.) We want the system to be able to employ the full expressive power of the first-order predicate calculus, including the ability to reason with disjunctive assertions, negations, and quantification of variables. Certainly the system should be sound (i.e., it should not prove invalid expressions). With regard to completeness (i.e., being able to prove invalid any theorem), we are less doctrinaire. We insist only that it behave reasonably according to criteria specific to the domain of application. Any incompletenesses that cannot be tolerated must be repairable by evolutionary changes to the system.

The classical model of theorem proving in the predicate calculus involves three major components. First, there is a set of axioms or assertions that express information about the domain of application. For geometry, for example, these would be the fundamental postulates plus whatever other theorems we want to start with. (It is neither necessary nor desirable to limit the assertions to some primitive or minimal set.) Second, there are domain-independent, uniform rules of inference (such as resolution, modus ponens) that can be used to derive new assertions from existing ones. Finally, there is a conjectured theorem, or goal, to be proved. A proof consists of a sequence of inference rule applications ending with one that produces the goal.

AI research has produced an important deviation from this approach. The assertions are divided into two distinct sets: <u>facts</u> and <u>rules</u>. Facts are specific statements about the particular problem at hand. For example, "Triangle ABC is a right triangle" would be expressed as a fact. Rules are general statements, usually involving implications or quantified variables. For example, "The base angles of an isoceles triangle are equal" would be expressed as a rule. Rules are used in combination with facts to produce derived facts. One could think of them as specialized, domain-dependent inference rules.

This distinction can be further explained by a simple example. In the classical approach, from the two assertions A and A=>B we could derive the assertion B by modus ponens. In the AI approach, from the fact A we could derive the fact B by using the special rule A=>B. The distinction between facts and rules is an important part of our deduction system.

The rules will be used as production rules. They will be invoked by a pattern matching process. Some will be used only in a <u>forward</u> direction for converting facts to derived facts; others will be used only in a <u>backward</u> direction for converting goals to subgoals. The developing sets of facts and goals will be represented by separate tree structures. Goals will be represented in an AND/OR goal tree, and facts will be represented in a newly proposed structure that we shall call a <u>fact tree</u>. Rules are employed until the fact tree joins the goal tree in an appropriate manner. The entire process will be under the supervision of a <u>control strategy</u> that decides which applicable rule should be employed at any stage. We shall not propose any specific control strategies in this paper but shall merely point out that the designer has the freedom to use any domain-specific information whatsoever in the control system.

Several designs of this general sort have been proposed (see, for example, Kowalski, op.cit.), but most of them have had restrictions on the kind of logical expressions that could be accommodated. Although AND/OR goal trees have been used before, the notion of a fact tree, dual to the goal tree, allows some interesting correspondences, such as that between "reasoning by cases" and dealing with conjunctive goals, for example.

It is hoped that the proposed system will serve as the beginning of a theoretical foundation for the various applications of "rule-based systems" now begin developed by AI research. Many of these systems are fundamentally deduction systems even though some of them allow uncertain or probabilistic facts and rules. Extending the present system so that it could also deal with uncertain knowledge would be a valuable future project.

3.4 Signal Understanding Systems

3.4.1 HPP-77-2

Title: A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data

Authors: Robert S. Engelmore and H. Penny Nii

A description of the problem of protein crystal structure inference is presented here, emphasizing those aspects of the analysis which are primarily non-numerical. The problem is one that requires the integration of multiple sources of knowledge, wherein the expert problem solver must interpret his experimental data in the light of general principles of protein chemistry, stereo-chemical constraints, specific details of the problem at hand, and heuristic problem solving strategies. A characteristic feature of the problem solving process is that one knowledge source enables other KSs to build further.

In designing a system to infer protein structures from x-ray crystallographic and other physical data, it was necessary to consider the appropriateness of other knowledge based system designs. The knowledge organization and control structures employed in Heuristic DENDRAL, MYCIN, HEARSAY II and HASP were considered. The use of a global knowledge base, as employed in HEARSAY II, and the event-driven control structure of HASP were found to be design elements well suited to the protein structure inference system (now called CRYSALIS).

3.4.2 HPP-77-7

Title: Rule-based Understanding of Signals

Authors: H. Penny Nii and Edward A. Feigenbaum

Knowledge-based programs which employ pattern-invoked inference methods were designed and at least partially implemented in two task domains. Both tasks are concerned with the interpretation of large quantities of digitized signal data. The task of SU/X is to understand "continuous signals," that is, signals which persist over time. The task of SU/P is to interpret protein x-ray crystallographic data. Some features of the design are: (1) incremental interpretation of data employing many different pattern-invoked sources of knowledge, (2) production rule representation of knowledge, including high level strategy model-driven techniques within a general hypothesize-and-test paradigm; and (4) multilevel representation of the solution hypothesis.

SU/X and SU/P are two application programs that have been written to reason toward an understanding of digitized physical signals. The essential features of the programs' design are: (1) data- and model-driven, opportunistic modes of hypothesis formation in which the "control" is organized hierarchically, and (2) a globally accessible hypothesis structure augmented by focus-of-attention and historical information which serve to integrate diverse sources of knowledge. The basic design is similar in many ways to the HEARSAY-II Speech Understanding System design. It is applicable to many different types of problems, especially to those problems that do not have computationally feasible "legal move generators" and must therefore resort to opportunistic generation of alternate hypotheses.

The use of production rules to represent control/strategy knowledge offers the advantages of uniformity of representation and accessibility of knowledge for purposes of augmentation and modification of the knowledge base. Because the line-of-reasoning is often a complex compounding of the elemental steps indicated by the rules, a dynamic explanation capability is needed.

3.5 Discovery as Heuristic Search

3.5.1 HPP-76-8

Title: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search

Author: Douglas Lenat

In his Ph.D. thesis, Lenat describes a program, called "AM", which models one aspect of elementary mathematics research: developing new concepts under the guidance of a large body of heuristic rules. "Mathematics" is considered as a type of intelligent behavior, not as a finished product.

This thesis is concerned with creative theory formation in mathematics: how to propose interesting new concepts and plausible hypotheses connecting them. The experimental vehicle of the research is a computer program called AM. Initially, AM is given the definitions of 115 simple set-theoretic concepts (like "Delete", "Equality"). Each concept is represented internally as a data structure with a couple dozen slots or facets (like "Definition", "Examples", "Worth"). Initially, most facets of most concepts are blank, and AM uses a collection of 250 heuristics - plausible rules of thumb - for guidance, as it tries to fill in those blanks. Some heuristics are used to select which specific facet of which specific concept to explore next, while others are used to actually find some appropriate information about the chosen facet. Other rules prompt AM to notice simple relationships between known concepts, to define promising new concepts to investigate, and to estimate how interesting each concept is. The same heuristics are used both to suggest new directions for investigation, and to limit attention: both to sprout and to prune.

Each concept is represented as a frame-like structure with 25 different facets or slots. The types of facets include: Examples, Definitions, Generalizations, Domain/Range, Analogies, Interestingness, and many others. Modular representation of concepts provides a convenient scheme for organizing the heuristics; for example, the following strategy fits into the Examples facet of the Predicate concept: "If, empirically, 10 times as many elements fail some predicate P, as satisfy it, then some generalization (weakened version) of P might be more interesting than P." AM considers this suggestion after trying to fill examples of each predicate.

The particular mathematical domain in which AM operates depends upon the choice of initial concepts. Currently, AM begins with nothing but a scanty knowledge of concepts which Piaget might describe as prenumerical: Sets, substitutions, operations, equality, and so on. In particular, AM is not told anything about proof, single-valued functions, or numbers.

From this primitive basis, AM quickly "discovered" elementary numerical concepts (corresponding to those we refer to as natural numbers, multiplication, factors, and primes) and wandered around in the domain of elementary number theory. AM was not designed to prove anything, but it did conjecture many well-known relationships (e.g., the unique factorization theorem).

AM is forced to judge a priori the value of each new concept, to lose interest quickly in concepts which aren't going to develop into anything. Often, such judgments can only be based on hindsight. For similar reasons, AM has difficulty formulating new heuristics which are relevant to the new concepts it creates. Heuristics are often merely compiled hindsight. While AM's "approach" to empirical research may be used in other scientific domains, the main limitation (reliance on hindsight) will probably recur. This prevents AM from progressing indefinitely far on its own.

This ultimate limitation was reached. AM's performance degraded more and more as it progressed further away from its initial base of concepts. Nevertheless, AM demonstrated that selected aspects of *creative discovery in elementary mathematics* could be adequately represented as a heuristic search process. Actually constructing a computer model of this activity has provided an experimental vehicle for studying the dynamics of plausible empirical inference.

3.6 Planning in an Experimental Science

3.6.1 HPP-77-5

Title: A Review of Knowledge-Based Systems as a Basis for a Genetics Experiment Designing System

Authors: Mark Stefik and Nancy Martin

This report is a slightly revised version of Stefik's thesis proposal submitted in December 1976. It is proposed here that the appropriate assemblage of knowledge-based problem solving techniques can be applied to the task of planning. The objective is a program, called MOLGEN, which serves as a laboratory assistant to a molecular geneticist. The program is expected to perform experiment checking and experiment planning. Experiment checking involves the computer simulation of previously designed experiments. This means that a set of input samples would be defined and a specific sequence of laboratory steps would be given. The computer system would then simulate the sequence of transformations on the representations of the samples terminating finally with a set of new samples. These new samples can be compared with actual laboratory results as a test of the initial hypotheses or of the accuracy of the transformations in the knowledge base. Such a system would be used by the system designers for debugging the transformation knowledge base and by geneticists for comparing the predicted results from the MOLGEN system against actual laboratory experiments. The checking facility would also be used to compare alternate experimental designs before investing any laboratory effort. A more sophisticated task for the program is the designing of experiments. This means that the program would need to know of the strategies involved in building sequences of transformations. This strategy knowledge would be in addition to the legal moves of genetics and encompasses a broad range of knowledge including such things as plan sketches for various contexts, design cost heuristics which predict the costs of considering certain design options, and mechanisms for evaluating the relevance and specificity of laboratory transformations to the current problem.

A substantial part of the effort in creating a system capable of designing experiments as a laboratory assistant centers around the creation and maintenance of an extensive genetics knowledge base. This implies a number of system capabilities to facilitate knowledge acquisition, integration, and debugging which are discussed further in the report.

The report contains a detailed review of research in problem solving and planning, and several extant systems (MYCIN, TEIRESIAS, HEARSAY) which manage large knowledge bases. The thrust of this proposal is based on the contention that many of the ideas which have proved important for the acquisition and management of object knowledge may be extended to cover action and strategy knowledge as well.

Parallel to the schemata based rule knowledge base is the concept of expressing the dynamic knowledge of the problem solving process through schemata. This leads to the development of the concept of a planning network. This network provides a mechanism for expressing the problem solving state in terms of a small number of node types corresponding to basic problem solving steps used at all levels. The planning network idea combines and extends the best elements of HEARSAY's blackboard NOAH's procedural network, and schemata based representations.

3.6.2 HPP-77-19

Title: Knowledge Base Management for Experiment Planning

Authors: Nancy Martin, Peter Friedland, Jonathan King and Mark Stefik

This is a progress report on the MOLGEN system as of spring 1977. The various types of knowledge required for planning experiments in molecular genetics are discussed, followed by the presentation of a "schema system" for representing all these types uniformly. Each schema will be composed of slots with associated value or type specifications and attached procedures. All schemata will be organized in a generalization/specialization hierarchy similar to that of other systems. Associated with each schema will be a model which summarizes the ways in which the various slots have been filled. These schemata and their associated models provide the knowledge needed by management routines for acquiring and updating the knowledge base and for many problem solving tasks. The issues involved in representing knowledge in such a schema system are similar to the issues when the representation is a semantic net.

A novel aspect of the design of the MOLGEN knowledge base is the representation of all types of problem solving knowledge in a common formalism -- as instances of schemata. Procedural knowledge will be represented in such a way that the system can easily inspect any procedure. The schema system provides a mechanism for breaking procedures into component parts which can be addressed separately, and are thus accessible to the system. This aids acquisition and use during problem solving.

3.7 Knowledge-based Programmer's Assistant

3.7.1 HPP-77- (Thesis in preparation)

Title: Thesis

Author: Denny Brown

Knowledge-Based Programmer's Assistant

This research began as an attempt to apply the research paradigm of the Heuristic Programming Project to computer program debugging. The goals of the research were to provide an analysis of debugging which would be a contribution to the art of programming and to produce a performance program capable of providing sophisticated debugging assistance to programmers. The analysis and the implementation of the debugging system (DBSYS) were developed in concert.

Analysis of Debugging

Debugging is viewed as a three dimensional space. The first dimension divides debugging into three problem solving tasks: Detection, Diagnosis, and Correction. The Detection phase of debugging involves the process of deciding that a problem indeed exists in the program. The Diagnosis phase involves the process of determining and isolating the cause of the problem. The Correction phase involves the process of deciding what modification is to be made in the program and making it. Along the second dimension, errors are divided into four classes: Syntactic, Semantic, Logical, and Pragmatic. Briefly, a Syntactic error occurs when the text of the program violates a rule of the programming language's syntax. A Semantic error occurs when a syntactically correct program attempts to execute operations which violate the semantics of the language, or whose results are undefined. A Logical error occurs when a program which is syntactically and semantically correct gives results which are "wrong". A Pragmatic error occurs when a program runs correctly but violates some additional constraint, such as high cost. Along the third dimension, we divide errors into three classes: Criterion, Consistency, and Completeness. Criterion bugs are bugs which occur in a program because the programmer has an incomplete or incorrect idea of the correctness conditions of the program. Consistency bugs occur when a programmer knows what he wants done, and how to do it, but does not construct the program accordingly. Completeness bugs occur when a programmer knows what he wants done, but doesn't really know how to do it.

Design of the DBSYS debugging system

The design of DBSYS thus evolved into the design of three separable mechanisms. The Acquisition sub-system is designed to obtain from the programmer necessary information concerning what the program is supposed to do. The Detection sub-system is designed to determine that the program doesn't do what it is supposed to do. The purpose of the Diagnosis and Correction sub-system is to find and fix the cause of the problem.

Performance was the main concern in the design of the Diagnosis and Correction stage. According to the global design principles, the system should be potentially attractive to programmers in their normal mode of operation. Consequently, there was strong motivation to have the Diagnosis and Correction mechanism do a very complete job. The system is designed to do all of the following:

1) Determine the location of the cause of the Error(s) detected.

2) Determine a modification to the program which would eliminate the Error.

3) Modify the text of the program accordingly.

4) Repair the executing environment in order to continue the execution as if the Error had not occurred.

In order to accomplish these tasks, two corollary sets of problems were faced. The first set involved the cognitive, problem-solving aspect of the work, namely, deciding upon a plan of action. The second set of problems dealt with the systems programming necessary to carry out the plan. Solutions to these two sets of problems were seen as potentially quite different. Consequently, the Diagnosis and Correction stage was designed in two comingled parts. The cognitive component is a knowledge-based, production-rule oriented system. Some of the primitives of the rule system, however, are complex system functions to handle the details of gathering information and making modifications.

Current state of progress

The DBSYS system runs and debugs programs. As yet, there is no theoretical characterization of the class of errors which the system can handle. Whether the system can detect, diagnose, and/or correct a particular bug depends on: 1) The information provided by the programmer in his description of his program. 2) The current state of the knowledge base of rules, which is always growing. 3) The "closeness" of the program to being correct. (E.g. A double-bug, where two problems interact, can't be corrected. The rule base knows more about problems which occur in reasonably well-structured programs which are almost right.)

A doctoral thesis which describes this work is currently being written.

3.8 Distributed Computing

3.8.1 HPP-77-12

Title: The Contract Net: A Formalism For the Control of Distributed Problem Solving

Author: Reid G. Smith

The following short paper, presented at the 5th IJCAI, introduces the concept of the contract net. The research in this area is still at an early stage, but simulations have already shown the idea to be promising, and its use in the area of data-base management is under investigation (see below).

Distributed processing offers the potential for high speed, reliable computation, together with a means to handle applications that have a natural spatial distribution. In this paper, distributed processing is defined as processing that is characterized by physical decomposition of the processor into relatively independent processor nodes. Recent advances in LSI technology, expected to result in single silicon wafers with at least 10^6 active elements by 1981, indicate that it is reasonable to contemplate designs which incorporate large networks of single chip processor nodes.

In this paper we examine the control of problem solving in such an environment, where most information is local to a node, and relatively little information is shared by the complete network. Individual nodes correspond to "experts" which cooperate to complete a top level task (analogous to Lenat's "beings"). The distributed processor is thus to be composed of a network of "loosely-coupled", asynchronous nodes, with distributed executive control, a flexible interconnection mechanism, and a minimum of centralized, shared resources. This puts the emphasis on "coarse grain" parallelism, in which individual nodes are primarily involved with computation (large kernel tasks), pausing only occasionally to communicate with other nodes.

The CONTRACT NET represents a formalism in which to express the control of problem solving in a distributed processor architecture. Individual tasks are dealt with as <u>contracts</u>. A node that generates a task broadcasts its existence to the other nodes in the net as a <u>contract announcement</u>, and acts as the <u>contract manager</u> for the duration of that contract. <u>Bids</u> are then received from potential <u>contractors</u>, which are simply other nodes in the net. An <u>award</u> is made to one node which assumes responsibility for the execution of that contract. <u>Subcontracts</u> may be let in turn as warranted by task size or a requirement for special expertise or data not in the possession of the contractor. When a contract has been executed, a <u>report</u> is transmitted to the contract manager.

Contracts may be announced via general broadcast, limited broadcast, or point-to-point communications mechanisms, depending on information about relevant contractors available to the contract manager. If, for example, a manager has knowledge about the location of particular data, then its contract announcement will be directed to the node(s) believed to possess that data, so that the complete network is not needlessly involved.

Contracting effectively distributes control throughout the network, thus allowing for flexibility and reliability. Decisions about what to do next are made as a result of relatively local considerations, between pairs of processors, although the nature of the announcement-bid-award sequence maintains an adequate global context; that is, the decision to bid on a particular contract is made on the basis of local knowledge (the task being processed in the node contemplating a bid), and global knowledge (other current contract announcements). The formalism also incorporates two way links between nodes that share responsibility for tasks (managers and contractors). The failure of a contractor is therefore not fatal, since a manager can re-announce a contract and recover from the failure.

A node in the CONTRACT NET is composed of a <u>contract processor</u>, <u>management processor</u>, communications interface, and local memory. The contract processor is responsible for the applications-related computation of the node. The management processor is responsible for network communications, contract management, bidding, and the management of the node itself. Individual nodes are not designated a priori as contract managers or contractors. Any node can take on either role, and during the course of problem solving, a particular node normally takes on both roles simultaneously for different contracts.

A contract is represented as a record structure with the following fields: NAME - the name of the contract, NODE - the name of another processor node associated with the contract, PRIORITY - a description of the "importance" of the contract, TASK - a description of the task to be performed, RESULTS - a description of the results obtained, and SUBCONTRACTS - a pointer to the list of subcontracts that have been generated from the contract.

Contracts are divided into two classes in a node: those for which the node acts as contractor, and those for which it acts as manager. The node field of a contract is filled accordingly - with the name of the contract manager in the first case, and with the name of the contractor in the second case. Subcontracts waiting for service are held at the node that generated them, with an empty node field.

The priority description is used by a management processor to establish a partial order over contracts to be announced, and by potential contractors to order contracts for the purpose of bidding. Similar descriptions are also used to order contractors for the purpose of awards. The concept of priority thus must be generalized over simple integer descriptions to include such (layered) descriptions of potentially arbitrary complexity, which include both applications-related and architecture-related information.

A task description also contains two types of information: the local context in which the task is to be executed, and the applications software required to execute it. This information is passed when a contract is awarded. Depending on the task, the required global context may be passed with the award, or further contracts may be let to obtain it. Software passed to a node for execution of a particular contract is retained for future use, and its presence has a favorable effect on the future bids of that node.

A SAIL simulation has been constructed to test the formalism. It accepts simple applications programs, and maps them onto a simulated distributed processor with a variable number of nodes. The simulation is being used to determine the costs associated with the formalism, in terms of both processor and communications overhead, and the decrease in computation time that can be expected for various applications. Distributed heuristic search is presently being examined in this way, and alternatives in distributed deduction will be examined in the near future.

3.8.2 HPP-77-21

Title: Application of the Contract Net Protocol To Distributed Data Bases

Authors: Hector Garcia-Molina and Gio Wiederhold

One application of the contract net protocol is to distributed data bases. In a distributed data base system, the data base is physically partitioned over several computing facilities while allowing integrated access to the data. Each computing facility, or node, includes a process which is in charge of the management of some section of the data base. Each node is connected to the other nodes to allow the sharing of data. A model for a distributed data base, using the concept of a contract, was developed by Garcia-Molina and Wiederhold.

Processing in this node is defined as follows: The user enters his query at a network node and it becomes a contract for the local contract processor. The query does not necessarily contain *a priori* information giving the relation name or the node name where relevant relations reside. A query which only involves local data will be answered directly and the result will be sent to the user. Since we are mainly interested in data base networks with intensive local node activities, we expect that most of the queries will be of this type.

If the interaction cannot be completely processed locally, then it must involve data from other nodes. The first step is to obtain the necessary domain knowledge from other nodes. The internal schemas provide the knowledge required to translate the query into a set of precise machine executable commands. The local processor will extract from the query all the terms it does not understand and will broadcast a contract announcement containing these terms. Other nodes on the network will examine the terms and if they find any of them in one of their internal schemas, they will return the relevant entries, the location of the relation, and the protection information in the form of a bid.

When the contract manager receives the bids, he will use the information in them to complete the query analysis. Having all the necessary information, the manager can generate the contract awards. The nodes that receive awards may in turn generate subcontracts if necessary.

From the bidding information, the contract manager will be able to decide which nodes are best equipped to solve the query (i.e., what nodes can respond faster, etc.). The manager can even decide to try processing the query in multiple ways. He will then issue contract awards to several nodes and will accept the first result that arrives. An option available to the manager is to issue an award to the node that has the prime copy of a relation and also to the nodes that have the backup relations. Because each relation might be stored differently or because the computing loads at the nodes vary, one node will send the answer back before the others. This way the response time of the system can be improved. The contractor may also keep the bid information around for a time, in case other similar queries follow. This "working set" strategy has been suggested in [7].

The following example will illustrate the processing mechanism in the data base. Suppose a user asks the query "FIND SALARY WHERE EMPLOYEE-NAME = FRED" at node X which has no data on salaries. The node will broadcast a contract announcement containing the incomprehensible words "SALARY", "EMPLOYEE-NAME", and "FRED". It might receive the following bids: From node A : I can give you SALARY if you give me SOC-SECURITY-NUM. From node B : I can give you SALARY if you give me JOB-RATING. From node C : If you give me EMPLOYEE-NAME, I will give you SOC-SECURITY-NUM, POSITION, etc.

Now node X can initiate one solution: Send a contract award to node C to get the social security number and then using this number issue another contract to node A to find the requested salary. While this solution attempt is going on, node X can attempt another approach. It broadcasts a contract announcement containing the word JOB-RATING and gets the following bid:

From node D : I can give you JOB-RATING if you give me POSITION. With this information, node X can start up a parallel solution attempt: Issue an award to node C to get the employee's position, then one to D to get the job rating and finally one to B to get the desired answer.

3.8.3 HPP-77-27

Title: Overview and Bibliography of Distributed Data Bases

Author: Hector Garcia-Molina

Garcia-Molina has written an overview and compiled an annotated bibliography of distributed data bases. These data bases are classified along ten different dimensions. Current areas of research are also categorized.

3.9 General discussions and surveys of HPP research activities

3.9.1 HPP-74-3

Title: Scientific Theory Formation by Computer

Author: Bruce G. Buchanan

This paper is an instance of our effort to transfer the concepts and results of the HPP work in automatic theory formation to the scientific community. Buchanan discusses theory formation in the context of the Meta-DENDRAL program, discussing in detail the four major steps of data selection, data interpretation, rule generation, and rule modification. The conclusion states:

"Chemists have also found that the output of Meta-DENDRAL alone was extremely useful to them, for its meticulous determination of all the explanatory processes that they would consider "plausible." We further expect the program to contribute to this domain of science by suggesting refinements or generalizations that chemists find useful.

Meta-DENDRAL has at least one serious limitation, though, which is common to all current computer approaches to theory formation. The program appears hopelessly bound to a model of the task domain. The primitive terms and relations that the program uses to define its rule space must be explicitly supplied to the program in some form. In the limit, nothing really "new" can be discovered by such a program. This type of objection has been repeatedly raised against the possibility of machine intelligence, since the first intelligent programs were conceived. Yet the criticism is unsettling.

A serious possibility for pushing the objection back another level, at least, is to search a space of models for the "best" one in which to do theory formation at any one time. This possibility has not been explored, because this is a second-order investigation in a field where first-order investigations are just beginning to succeed. However, there is no reason to believe that model-building is theoretically impossible any more than rule-building is impossible.

3.9.2 HPP-74-4

Title: Computer Applications: Introductory Remarks

Author: Edward A. Feigenbaum

Most of this report is reproduced in Sect. 2.1 above.

3.9.3 HPP-75-4

Title: Judgmental Knowledge as a Basis for Computer-Assisted Clinical Decision Making

Author: Edward H. Shortliffe

The DENDRAL programs have for many years provided case studies in which various aspects of AI research were explored: generality vs specificity, representation of knowledge, heuristic search, etc.

Recently the programs also have served as a testing ground for exploring technology transfer through the development of a collaborative research community. In a contribution to the book, Computer Networking and Chemistry (edited by Peter Lykos), Carhart et al discusses many of the human engineering issues which were faced in making the DENDRAL programs acceptable to users outside the Stanford community.

3.9.4 HPP-75-13
Title: Applications of Artificial Intelligence to Scientific Reasoning

Author: Bruce G. Buchanan

This is an instance of our technology transfer efforts. The paper attempts to distill the knowledge derived from our successful transfer of AI programs, specifically, Heuristic DENDRAL and Meta-DENDRAL, from the research computer science laboratory to the chemistry laboratory. The desiderata of high performance, use of extensive task-specific knowledge, flexibility, and interactive programs are discussed. Other issues discussed in the paper include the use of the plan-generator test paradigm, the choice of the appropriate level of conceptualization, the rule of the expert, and the choice of programming language.

3.9.5 HPP-77-15
Title: A Correlation Between Crystallographic Computing and Artificial Intelligence

Authors: E. A. Feigenbaum, R. S. Engelmore, C. K. Johnson

This paper resulted from an invited talk given by Feigenbaum to the American Crystallographic Association in January, 1976. The paper appeared in Acta Crystallographica, a leading technical journal for crystallographers, and represents another effort to introduce the themes of AI research to scientists in various disciplines. A brief overview is given of some applied artificial intelligence research projects in scene analysis, medical consultation, mass-spectral analysis, knowledge acquisition, chemical synthesis and density-map interpretation. These examples are given to illustrate the thesis that symbolic computing techniques are indeed useful in scientific computing applications.

3.9.6 HPP-75-16
Title: Some Considerations for the Implementation of Knowledge-Based Expert Systems

Authors: E. H. Shortliffe and R. Davis

The transfer of a high performance AI application program to the target community involves considerably more than the implementation and debugging of the program. The authors define eight stages, or milestones, which a system must pass before it evolves to a real utility outside the computer science community. The fundamental point is that these systems should be designed with all eight milestones in mind, and introduced to their users carefully.

3.9.7 HPP-77-25
Title: The Art of Artificial Intelligence

Author: Edward A. Feigenbaum

In this invited paper, presented at the Fifth IJCAI, Feigenbaum examines emerging themes of knowledge engineering, illustrates them with case studies drawn from the work of the Stanford Heuristic Programming Project, and discusses general issues of knowledge engineering art and practice. The major themes are described as follows:

Generation-and-test: Omnipresent in our experiments is the "classical" generation-and-test framework that has been the hallmark of AI programs for two decades. This is not a consequence of a doctrinaire attitude on our part about heuristic search, but rather of the usefulness and sufficiency of the concept.

Situation = Action Rules>: We have chosen to represent the knowledge of experts in this form. Making no doctrinaire claims for the universal applicability of this representation, we nonetheless point to the demonstrated utility of the rule-based representation. From this representation flow rather directly many of the characteristics of our programs: for example, ease of modification of the knowledge, ease of explanation. The essence of our approach is that a rule must capture a "chunk" of domain knowledge that is meaningful, in and of itself, to the domain specialist. Thus our rules bear only a historical relationship to the production rules used by Newell and Simon (1972) which we view as "machine-language programming" of a recognize => act machine.

The Domain-Specific Knowledge: It plays a critical role in organizing and constraining search. The theme is that in the knowledge is the power. The interesting action arises from the knowledge base, not the inference engine. We use knowledge in rule form (discussed above), in the form of inferentially-rich models based on theory, and in the form of tableaus of symbolic data and relationships (i.e. frame-like structures). System processes are made to conform to natural and convenient representations of the domain-specific knowledge.

Flexibility to modify the knowledge base: If the so-called "grain size" of the knowledge representation is chosen properly (i.e. small enough to be comprehensible but large enough to be meaningful to the domain specialist), then the rule-based approach allows great flexibility for adding, removing, or changing knowledge in the system.

Line-of-reasoning: A central organizing principle in the design of knowledge-based intelligent agents is the maintenance of a line-of-reasoning that is comprehensible to the domain specialist. This principle is, of course, not a logical necessity, but seems to us to be an engineering principle of major importance.

Multiple Sources of Knowledge: The formation and maintenance (support) of the line-of-reasoning usually require the integration of many disparate sources of knowledge. The representational and inferential problems in achieving a smooth and effective integration are formidable engineering problems.

Explanation: The ability to explain the line-of-reasoning in a language convenient to the user is necessary for application and for system development (e.g. for debugging and for extending the knowledge base). Once again, this is an engineering principle, but very important. What constitutes "an explanation" is not a simple concept, and considerable thought needs to be given, in each case, to the structuring of explanations.

The case studies described in the remainder of the paper include DENDRAL, Meta-DENDRAL, MYCIN and TEIRESIAS, SU/X, AM, MOLGEN and CRYSALIS.

4 Bibliography

HPP-74-3
B.G. Buchanan,
"Scientific Theory Formation by Computer," Nato Advanced Study Institutes Series, Series E: Applied Science, 14:515, Noordhoff-Leyden, (1976).

HPP-74-4
E.A. Feigenbaum,
"Computer Applications: Introductory Remarks," in Proceedings of Federation of American Societies for Experimental Biology, 33:2331, (1974).

HPP-75-1
E.H. Shortliffe, B.G. Buchanan,
"A Model of Inexact Reasoning in Medicine," Mathematical Biosciences, 23:351, (1975).

HPP-75-2
E.H. Shortliffe, R. Davis, S.G. Axline, B.G. Buchanan, C.C. Green, S.N. Cohen,
"Computer-Based Consultations in Clinical Therapeutics; Explanation and Rule Acquisition Capabilities of the MYCIN System," Computers and Biomedical Research, 8:303, (August 1975).

HPP-75-4
E.H. Shortliffe,
"Judgmental Knowledge as a Basis for Computer-Assisted Clinical Decision Making," Proceedings of the 1975 International Conference on Cybernetics and Society, (September 1975).

HPP-75-6
AIM-266, CS-517, ADA019641
Randall Davis, Bruce Buchanan, Edward Shortliffe,
"Production Rules as a Representation of a Knowledge-Based Consultation Program," (October 1975). Also found in Artificial Intelligence, 8:1(February 1977).

HPP-75-7
AIM-271, CS-524,, ADA019702/OWC
Randall Davis, Jonathan King,
"An Overview of Production Systems," Machine Representations of Knowledge, Dordrecht, D. Reidel Publ. Co., 1976. Proceedings of NATO ASI Conference, (October 1975).

HPP-75-13
B.G. Buchanan,
"Applications of Artificial Intelligence to Scientific Reasoning," In Proceedings of Second USA-Japan Computer Conference, American Federation of Information Processing Societies Press, (August 1975).

HPP-75-14
R.E. Carhart, S.M. Johnson, D.H. Smith, B.G. Buchanan, R.G Dromey, J. Lederberg,
"Networking and a Collaborative Research Community: A Case Study Using the DENDRAL Program," in "Computer Networking and Chemistry", P. Lykos, Ed., American Chemistry Society, Washington, D.C., (1975).

HPP-75-16
E.H. Shortliffe, R. Davis,
"Some Considerations for the Implementation of Knowledge-Based Expert Systems," SIGART Newsletter, 55:9, (December 1975).

HPP-76-4
B.G. Buchanan, D.H. Smith, W.C. White, R.J. Gritter, E.A. Feigenbaum, J. Lederberg, and Carl Djerassi,
"Application of Artificial Intelligence for Chemical Inference XXII. Automatic Rule Formation in Mass Spectrometry by Means of the Meta-DENDRAL Program," Journal of the American Chemical Society, 98:6168, (1976).

HPP-76-5
T.H. Varkony, R.E. Carhart and D.H. Smith,
"Applications of Artificial Intelligence for Chemical Inference XXIII. Computer-Assisted Structure Elucidation. Modelling Chemical Reaction Sequences Used in Molecular Structure Problems," in "Computer-Assisted Organic Synthesis," W.T. Wipke, Ed., American Chemical Society, Washington, D.C., (in press).

HPP-76-7
AIM-283, CS-552
Randall Davis,
"Applications of Meta Level Knowledge to the Construction, Maintenance and Use of Large Knowledge Bases,"
Ph.D. Thesis in Computer Science, (July 1976).

HPP-76-8
AIM-286, CS-570
Douglas Lenat,
"AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search,"
Ph.D. Thesis in Computer Science, (July 1976).

HPP-77-1
STAN-CS-77-593
A.C. Scott, W. Clancey, R. Davis, E.H. Shortliffe,
"Explanation Capabilities of Knowledge-Based Production Systems," American Journal of Computational Linguistics, Microfiche 62, Knowledge-Based Consultation Systems, (1977).

HPP-77-2
STAN-CS-77-589
Robert S. Engelmore and H. Penny Nii
"A Knowledge-Based System for the Interpretation of Protein X-Ray Crystallographic Data," (January 1977).

HPP-77-4
T.M. Mitchell and G.M. Schwenzer,
"Applications of Artificial Intelligence for Chemical Inference XXV. A Computer Program For Automated Empirical 13C NMR Rule Formation," (Submitted to Organic Magnetic Resonance, January 1977).

HPP-77-5
STAN-CS-77-596
Mark Stefik and Nancy Martin,
"A Review of Knowledge-Based Systems as a Basis for a Genetics Experiment Designing System," (February 1977).

HPP-77-6
STAN-CS-77-597
Bruce G. Buchanan and Tom Mitchell.
"Model-Directed Learning of Production Rules," in Proceedings for the Workshop on Pattern-Directed Inference Systems (February, 1977).

HPP-77-7
STAN-CS-77-612
H. Penny Nii and Edward A. Feigenbaum,
"Rule-based Understanding of Signals," to be presented at Workshop on Pattern-directed Inference Systems, (May 1977).

HPP-77-8
R. Davis,
"Knowledge Acquisition on Rule-based Systems: Knowledge about Representations as a Basis for System Construction and Maintenance," (Submitted to Pattern-Directed Inference Conference, May 1977).

HPP-77-9
R. Davis,
"Interactive Transfer of Expertise I: Acquisition of New Inference Rules," (Submitted to Fifth IJCAI, August 1977).

HPP-77-12
Reid G. Smith,
"The Contract Net: A Formalism For the Control of Distributed Problem Solving,"
(Submitted to Fifth IJCAI, August 1977).

HPP-77-13
Tom M. Mitchell,
"Version Spaces: An Approach to Rule Revision During
Rule Induction," (Proceedings of the Fifth IJCAI, August 1977).

HPP-77-14
Reid G. Smith, Tom M. Mitchell, Richard A. Chestek,
and Bruce G. Buchanan,
"A Model For Learning Systems," (Submitted to Fifth IJCAI,
August 1977).

HPP-77-15
E.A. Feigenbaum, R.S. Engelmore, C.K. Johnson,
"A Correlation Between Crystallographic Computing and
Artificial Intelligence," in Acta Crystallographica,
A33:13, (1977).

HPP-77-16
Randall Davis and Bruce G. Buchanan,
"Meta-Level Knowledge: Overview and Applications,"
(Submitted to Fifth IJCAI, August 1977).

HPP-77-19
N. Martin, P. Friedland, J. King, and M.J. Stefik,
"Knowledge Base Management for Experiment Planning,"
(submitted to 5th IJCAI).

HPP-77-25
Edward A. Feigenbaum,
"The Art of Artificial Intelligence," to appear in
the proceedings of 5th IJCAI, 1977.

HPP-77-27
Hector Garcia-Molina,
Overview and Bibliography of Distributed Data Bases, 1977.

HPP-77-28
STAN-CS-618
Nils J. Nilsson,
A Production System for Automatic Deduction,
Machine Intelligence 9, 1977.

HPP-77-33
Randall Davis,
Generalized Procedure Calling and Content-Directed Invocation,
Proceedings of Artificial Intelligence and Programming Languages Conf.,
Published as SIGART/SIGPLAN Combined Issue, August 1977, pp 45-54.

5 Appendix -- The AI Handbook

5.1 Introduction

The AI Handbook is a compendium of short articles (3-5 pages) about the projects, ideas, problems and techniques that make up the field of Artificial Intelligence. Over 150 articles have been drafted by researchers and students in the field, on topics ranging in depth from "ATN's" to "An Overview of Natural Language Research", and covering the entire breadth of AI research: search, robotics, speech understanding, real-world applications, etc.

Current work involves editing and rewriting of articles in all fields, as well as continuing research and article preparation in the fields of Theorem Proving, Vision, Robotics, Human Information Processing, Learning and Inductive Inference, Problem Solving, and Planning.

5.2 Table of Contents

This is a tentative outline of the Handbook. Articles in the first eight Chapters are expected to appear in the first volume. A list of the articles in each Chapter is appended.

II. Heuristic Search
- A. Overview
- B. Problem representation
  1. Overview
  2. State space representation
  3. Problem reduction representation
  4. Game trees
- C. Search
  1. Blind state space search
  2. Blind search of an and/or tree
  3. Heuristic search in problem-solving
     - a. Basic concepts
     - b. A* (optimal state-space search)
     - c. Variations on A*: band width, bidirectional search
     - d. Heuristic search of an and/or graph
     - e. Hill Climbing
  4. Game tree search
     - a. Minimax
     - b. Alpha-beta
- D. Examples
  1. Logic Theorist
  2. GPS
  3. Gelernter - geometry
  4. Slagle & Moses - Integration
  5. STRIPS
  6. ABSTRIPS
  7. NOAH

III. AI Languages
- A. Early list-processing languages
- B. Language/system features
  0. Overview of current LP languages
  1. Control structures
  2. Data Structures (lists, associations,
  3. Pattern Matching in AI languages
  4. Deductive mechanisms
- C. Current languages/systems
  1. LISP, the basic idea
  2. INTERLISP
  3. QLISP (mention QA4)
  4. SAIL/LEAP
  5. PLANNER
  6. CONNIVER
  7. SLIP
  8. POP-2
  9. SNOBOL
  10. QA3/PROLOGUE
  11. ACTORS, SMALLTALK, SIMULA

IV. Representation of Knowledge

- A. Overviews
  - 1. Survey of representation techniques
  - 2. Issues and problems in representation theory
- B. Representation Schemes
  - 1. Predicate calculus
  - 2. Semantic nets (Quillian, LNR, Hendrix)
  - 3. Production systems
  - 4. Higher Level Knowledge Structures
    - a. Frames, Scripts, The basic idea (Bartlett, Minsky, Abelson)
    - b. KRL-0, MERLIN
    - c. Others: FRL, OWL, Toronto
  - 5. Componential analysis (Schank, Wilks, Jackendoff)
  - 6. Procedural representations (SHRDLU, actors, demons)
  - 7. Direct (Analogical) representations
  - 8. Multiple Knowledge sources - Blackboard (see VIA)
- C. Comparison of Representation Schemes

V. Natural Language

- A. Overview - History & Issues
- B. Representation of Meaning
- C. Grammars and Parsing
  - 1. Review of formal grammars
  - 2. Extended grammars
    - a. Transformational grammars
    - b. Systemic grammars
    - c. Case Grammars
  - 3. Parsing techniques
    - a. Overview of parsing techniques
    - b. Augmented transition nets, Woods
    - c. CHARTS - GSP
- D. Text Generating systems
- E. Machine Translation
  - 1. Overview & history
  - 2. Wilks' machine translation work
- F. Famous Natural Language systems
  - 1. Early NL systems (SAD-SAM through ELIZA)
  - 2. PARRY
  - 3. MARGIE
  - 4. LUNAR
  - 5. SHRDLU, Winograd

VI. SPEECH UNDERSTANDING SYSTEMS

- A. Overview (include a mention of acoustic proc.)
- B. Design Considerations for Speech Systems
- C. Integration of Multiple Sources of Knowledge (see Overview)
- D. The Early ARPA speech systems
  - 1. DRAGON
  - 2. HEARSAY I

IX. THEOREM PROVING
- A. Overview
- B. Predicate Calculus
- C. Resolution Theorem Proving
  1. Basic resolution method
  2. Syntactic ordering strategies
  3. Semantic & syntactic refinement
- D. Non-resolution theorem proving
  0. Overview
  1. Natural deduction
  2. Boyer-Moore
  3. LCF
- E. Uses of theorem proving
  1. Use in question answering
  [2. Use in problem solving]
  3. Theorem Proving languages
  4. Man-machine theorem proving
  5. In Automatic Programming
- F. Proof checkers

X. VISION
- A. Overview
- B. Polyhedral or Blocks World Vision
  1. Overview
  2. Guzman
  3. Falk
  4. Waltz
- C. Scene Analysis
  1. Overview
  2. Template Matching
  3. Edge Detection
  4. Homogeneous Coordinates
  5. Line Description
  6. Noise Removal
  7. Shape Description
  8. Region Growing (Yakamovsky, Olander)
  9. Contour Following
  10. Spatial Filtering
  11. Front End Particulars
  12. Syntactic Methods
  13. Descriptive Methods
- D. Robot and Industrial Vision Systems
  1. Overview and State of the Art
  2. Hardware
- E. Pattern Recognition
  1. Overview
  2. Statistical Methods and Applications
  3. Descriptive Methods and Applications
- F. Miscellaneous
  1. Multisensory Images

2. Perceptrons

XI. ROBOTICS
A. Overview
B. Robot Planning and Problem Solving
C. Arms
D. Present Day Industrial Robots
E. Robotics Programming Languages

XII. Human Information Processing -- Psychology
A. Perception
B. Memory and Learning
1. Basic structures and processes in IPP
2. Memory Models
a. semantic net memory models
b. HAM (Anderson & Bower)
c. EPAM
d. Productions (HPS)
e. Conceptual Dependency
C. Psycholinguistics
D. Human Problem Solving
0. Overview
1. PBG's
2. Human chess problem solving
E. Behavioral Modeling
1. Belief Systems
2. Conversational Postulates (Grice, TW)
3. PARRY

XIII. Learning and Inductive Inference
A. Overview
B. Samuel Checker program
C. Winston -- concept formation
D. Pattern extrapolation problems--Simon,
E. Overview of Induction
F. AQVAL (Michalski at U.Ill)
G. Parameter adjustment of linear functions
H. Rote learning
I. D.A. Waterman's machine learning of heuristics
J. Learning by debugging
K. Learning by parameter Adaptation
L. Signature & move phase tables

XIV. Problem Solving & Planning
A. Overview
B. Problem Representation (See Section IIB)
C. Search (See Chapter II)
D. Theorem proving in problem solving

E. Planning
F. Constraint relaxation (Waltz, REF-ARF)
G. Reasoning by analogy (Evans, Zorba, Winston)
H. Problem Solving Programs
   1. STRIPS (See IID5)
   2. ABSTRIPS (See IID6)
   3. NOAH (See IID7)
   4. ZORBA
   5. QA3
   6. Evans analogy program

5.3 Sample Articles

5.3.1 HEURISTIC SEARCH OVERVIEW (Sect. II.A)

BASIC CONCEPTS IN HEURISTIC SEARCH

In the blind search of a state space (section IIC1) or and/or graph (section IIC2), the number of nodes expanded before reaching a solution is likely to be prohibitively large. Usually, one runs out of space or time (or both) in any but the simplest problems. This is because the order of expanding the nodes is purely arbitrary, and does not use any properties of the problem being solved. Information about the particular problem domain can often be brought to bear to help reduce the search. Information of this sort is called heuristic information, and a search method using it is a heuristic search method.

THE IMPORTANCE OF HEURISTIC SEARCH THEORY

Heuristic search methods were employed by nearly all early problem-solving programs. Most of these programs, though, were written to solve problems from only a single domain, and the domain-specific information they used was closely intertwined with the techniques for using it. This meant that the heuristic techniques themselves were not easily accessible for study and adaptation to new problems, and there was some likelihood that substantially similar techniques would have to be reinvented repeatedly. Consequently, an interest arose in developing generalized heuristic search algorithms, whose properties could be studied independently of the particular programs that might use them. (See Newell and Ernst, 1969; Feigenbaum 1969; Sandewall 1971.) This task, in turn, required the use of generalized problem formulations. The latter have been discussed in section ???, Problem Representation, in an approach generally following Nilsson (1971). Given a generalized problem representation, the most basic heuristic search techniques can be studied as variations on blind search for the same type of problem representation.

The importance of studying heuristic search algorithms in the abstract has been diversely judged in recent years. One of the best known students of the subject, has remarked, "The problem of efficiently searching a graph has essentially been solved and thus no longer occupies AI researchers" (Nilsson, 1974). Nevertheless, recent work makes it clear that heuristic search theory is far from complete (e.g., Gaschnig, 1977; Simon and Kadane, 1975); its kinship with complexity theory now tends to be emphasized (see Pohl, 1977).

WAYS OF USING HEURISTIC INFORMATION

The points at which heuristic information can be applied in a search include

(1) deciding which node to expand next, instead of doing the expansions in a strictly breadth-first or depth-first order, and

(2) in the course of expanding a node, deciding which successor or successors to generate, instead of blindly generating all possible successors at one time.

Decisions of the second type may often be identified with deciding which operator to apply next to a given node. A node to which some but not all applicable operators have been applied is said to have been partially developed or partially expanded. The use of heuristic information to develop nodes partially, reserving the possibility of fuller expansion at a later point in the search, has been investigated in Michie, 1967, and Michie and Ross, 1970. Other applications of the idea of limiting the successors of a given node occur in game-playing programs (see section ???, Game Tree Search). An important variant of the idea is means-ends analysis, which, instead of deciding on an applicable operator, chooses an operator most likely to advance the search whether or not it is immediately applicable. The problem of making the operator applicable, if necessary, is addressed secondarily. (See section ???, GPS, and section ???, STRIPS.)

Most theoretical study has concerned decisions of the first type, deciding which node to expand next, with the assumption that nodes are to be expanded fully or not at all. The general idea is always to choose for expansion the node that seems "most promising." The promise of a node can be defined in various ways. One way, in a state-space problem, is to estimate its distance from a goal node; another is to assume the solution path includes the node being evaluated and to estimate the length or difficulty of the entire path. Along a different dimension, the evaluation may take into account only certain predetermined features of the node in question, or it may determine the relevant features by comparing the given node with the goal. In all these cases, the measure by which the promise of a node is estimated is called an evaluation function. [Third sentence of paragraph is based on discussion in Human Problem Solving.] [Sandewall mentions a third possible way to evaluate promise of a node: try to estimate total remaining search effort at the node. Probably omit this.]

ORDERED STATE-SPACE SEARCH

An example of a heuristic search algorithm using an evaluation function is the ordered search of a state space. The evaluation function is f*; it is defined so that the more promising a node is, the smaller is the value of f*. The node selected for expansion is one at which f* is minimum.

The following algorithm describes an ordered search for a general state-space graph.

1. Put the start node s in a list, called OPEN, of unexpanded nodes. Calculate f*(s), and associate its value with node s.

2. If OPEN is empty, no solution exists.

3. Select from OPEN a node i at which f* is minimum. If several nodes qualify, choose among them arbitrarily (unless one or more is a goal node, in which case choose one of the goal nodes).

4. Remove node i from OPEN, and place it on a list, called CLOSED, of expanded nodes.

5. If i is a goal node, a solution has been found.

6. Expand node i, creating nodes for all its successors. For every successor node j of i:

   a. If j is neither in list OPEN nor in CLOSED, then add it to OPEN. Calculate f*(j) and associate its value with node j. Create a back pointer from j to its predecessor i (in order to trace back a solution path once a goal node is found).

   b. If j was already on either OPEN or CLOSED, calculate f*(j) and associate with j the smaller of the values f*(j) and the previous value associated with it.

   If the value associated with j has been lowered, change the back pointer so that it points to i. If j was in CLOSED and had its value lowered, move it back to OPEN.

   (This step is necessary for general graphs, in which a node can have more than one predecessor. The predecessor yielding the smaller value of f*(j) is chosen. For trees, in which a node has at most one predecessor, this step can be ignored. Note that even if the search space is a general graph, the subgraph that is made explicit is always a tree since node j never records more than one predecessor at a time.)

7. Go to 2.

Breadth-first, uniform-cost, and depth-first search (section ???, Blind State-Space Search) are all special cases of the ordered search technique. For breadth-first search, we choose f*(i) to be the depth of node i. For uniform-cost search, f*(i) is the cost of the path from the start node to node i. A depth-first search (without a depth bound) can be obtained by taking f*(i) to be the negative of the depth of the node.

The purpose of ordered search, of course, is to reduce the number of nodes expanded in comparison to blind search algorithms. Its effectiveness in doing so depends directly on the choice of f* as a means of discriminating sharply between promising and unpromising nodes. If the discrimination is inaccurate, however, the ordered search may miss an optimal solution or all solutions. If no exact measure of promise is available, the choice of f* thus involves a tradeoff between time and space on the one hand and the guarantee of an optimal solution, or any solution, on the other.

PROBLEM TYPES AND THE CHOICE OF F*

What it means for a node to be promising--and consequently, the appropriateness of a particular evaluation function--depends on the problem at hand. Several cases can be distinguished according to the type of solution required. In one, it is assumed that the state space contains multiple solution paths with different costs; the problem is to find an optimal (i.e., minimum cost) solution. This first case is well understood; see section ???, A*.

The second situation is similar to the first, but with an added condition: the problem is hard enough that, if it is treated as an instance of case one, the search will probably exceed time and space bounds without finding a solution. The questions for case two thus involve ways of finding good (but not optimal) solutions with reasonable amounts of search effort; and of bounding both the search effort and the extent to which the solution produced is less than optimal. For discussion of case two, see section ???, Relaxing the Optimality Requirement.

A third kind of problem is one in which there is no concern for the optimality of the solution; perhaps only one solution exists, or any solution is as good as any other. The question here is how to minimize the search effort, instead of, as in case two, trying to minimize some combination of search effort and solution cost. A probabilistic analysis has recently been given by Simon and Kadane (1975).

An example of case three comes from theorem-proving, where one may well be satisfied with the most easily found proof, however inelegant. A clear example of case two is the traveling-salesman problem (see section ???), in which finding some circuit through a set of cities is trivial and the difficulty, which is very great, is entirely in finding a shortest or close-to-shortest path. Most treatments, however, do not clearly distinguish between the two cases. A popular test problem, the 8-puzzle (see section ???), can be interpreted as being in either class.

RELATED ARTICLES:

State space representation
Search applied to state spaces
Evaluation functions

REFERENCES:

Allen Newell and George Ernst, "The Search for Generality," in Wayne A. Kalenich, ed., Information Processing 1965: Proceedings of IFIP Congress 65, Spartan Books, Washington, 1965, pp. 17-24.

Edward A. Feigenbaum, "Artificial Intelligence: Themes in the Second Decade," in A.J.H. Morrell, ed., Information Processing 68: Proceedings of IFIP Congress 1968, Volume 2, North-Holland, Amsterdam, 1969, pp. 1008-1024.

E. J. Sandewall, "Heuristic Search: Concepts and Methods," in N. V. Findler and Bernard Meltzer, eds., Artificial Intelligence and Heuristic Programming, American Elsevier, New York, 1971, pp. 81-100.

Peter E. Hart, Nils J. Nilsson, and Bertram Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," IEEE Transactions on Systems Science and Cybernetics, vol. SSC-4, 1968, pp. 100-107.

Nils J. Nilsson, Problem-Solving Methods in Artificial Intelligence, McGraw-Hill, New York, 1971.

D. Michie, "Strategy Building with the Graph Traverser," in N. L. Collins and Donald Michie, eds., Machine Intelligence 1, American Elsevier, New York, 1967, pp. 135-152.

Donald Michie and Robert Ross, "Experiments with the Adaptive Graph Traverser," Bernard Meltzer and Donald Michie, eds., Machine Intelligence 5, American Elsevier, New York, 1970, pp. 301-318.

5.3.2 NATURAL LANGUAGE PROCESSING OVERVIEW (Sect. V.A)

Prehistory

Computational linguistics, the precursor of current work in machine understanding of natural languages, first appeared in the 1940s, soon after computers became available commercially. The machine's ability to manipulate symbols was first used to compile lists of word occurrences (word lists) and concordances (their contexts in written texts). Such surface-level machine processing of text was of some value in linguistic research, but it soon became apparent that the computer could perform much more powerful linguistic functions than merely counting and rearranging data.

In 1949, Warren Weaver proposed that computers might be useful for "the solution of the world-wide translation problem" (Weaver, 1949, p. 15). The resulting research effort, called mechanical translation (see Article IE1), attempted to simulate with a computer the presumed functions of a human translator: looking up each word in a bilingual dictionary; choosing an equivalent word in the output language; and, after processing each sentence, arranging the resulting string of words to fit the output language's word order. Despite the attractive simplicity of the idea, many unforseen problems arose, both in selecting appropriate word equivalences and in arranging them to produce a sentence in the output language.

Goals

In the 1960s a new breed of computer programs was developed that attempted to deal with some of the more complex issues of language that had led to the difficulties in the mechanical translation efforts (see Article IF1). The central assumption behind these programs, which were the beginnings of natural-language-processing research, was that human communication is not a simple process of word manipulation. Rather, linguistic and psychological research indicates that human language is a complex cognitive ability involving many different kinds of knowledge: the structure of sentences; the meaning of words; a model of the listener; the rules of conversation; and an extensive, shared body of general information about the world. The focus of modern work in natural language processing is "understanding" language, defined in terms of performing tasks like paraphrasing, question answering, or information retrieval. The general approach has been to model human language as a knowledge-based communication processing system and then to create a working model of this processor in a digital computer.

In the field, researchers experiment with computer models of language processing not only to create useful "understanding" systems but to gain a better understanding of language, in general, and of the nature of computer intelligence, in particular. Like the human mind, the computer has the ability to manipulate symbols in complex processes, including processes that involve decision making based on stored knowledge. (It is an assumption of the field that the human use of language is a cognitive process of this sort.) By developing and testing computer-based models of language processing that approximate human performance, researchers hope to see more clearly how human language works. At the same time, it is hoped that such programs will be able to perform practical, language understanding tasks.

History

Natural language programs have had diverse methods and goals, making their categorization somewhat difficult. One coherent schema, borrowed from Winograd, 1972, groups natural-language programs according to how they represent and use knowledge of their subject matter. On this basis, natural language programs can be divided into four historical categories.

Early natural language programs worked with special formats to achieve limited results in specified domains. These programs used ad hoc data structures to represent "knowledge." Programs such as BASEBALL, SAD-SAM, STUDENT, and ELIZA (see Article IF1) searched their input sentences, which were restricted to simple declarative and interrogative forms, for key words or patterns representing known relationships and applied to them domain-specific heuristics and knowledge. Though they performed relatively small tasks and avoided or ignored complexities in language, their results and methods were the impetus to dealing with these more difficult problems.

The second category can be called text-based systems. These programs, such as PROTOSYNTHEX I (Simon et al., 1966) and the Teachable Language Comprehender (TLC) (Quillian, 1968), attempted to expand beyond the limits of a specific domain. These programs dealt with full English text as a base, rather than with key words or phrases. Input text was interpreted as a request to access a structured information store, and a variety of clever methods was used to identify the proper response. Though more general than their predecessors, these programs still failed to deal with the underlying meaning of the English language input. They were able to give only responses that had been pre-stored as data; they had no deductive power.

To try to deal with the problem of how to characterize and use the meaning of sentences, a group of programs was developed called limited logic systems. In these systems (e.g., SIR [Raphael,1968], DECON [Thompson,1968], and CONVERSE [Kellogg,1968]), the information in the database is stored in a formal, albeit ad hoc, notation, and mechanisms are provided for translating input sentences into the same form. This formal notation tries to free the informational content of the input from the structure of English. The overall goal of these systems was to accept complex input information (e.g., information containing quantifiers and relationships), use it to perform inferences on the database, and thus realize answers to complex questions. Their problem lay in the fact that the complexity of the stored information was not really part of the database but was built into the system's routines for manipulating the database. PROTOSYNTHEX II, for example, contained statements of the form "A is X" and "X is B" and tried to answer "Is A B?", based on transitivity. The deductive mechanism required for these inferences was embedded in special-purpose subroutines, rather than in the database as a "theorem," and thus was not available to be used to perform more involved inferences, which require a longer chain of reasoning.

More recently, natural language programs have leaned toward developing general deductive systems like those of Woods (1968) and Hewitt (1969, 1971) that attempt to express knowledge in an integrated format (see Article IF3; Article IF4; Article IF5; Article IE2; and Section , on specific systems). These systems perform much better than earlier ones, through the use of linguistic models of sentence structure and general reasoning techniques that operate on the meaning extracted from sentences, which is stored in some formal representation.

Concepts and Representation

Ultimately, natural language programs seek to match the performance of a human, language user. They attempt to simulate user functions through a variety of schemes for representation of knowledge and manipulation (see Section). These programs have, of course, been developed with very little understanding of the internal codes and processes of the human system being modeled.

In order to deal with the infinite complexities of a language in a systematic way, natural language programs constrain their input within the framework of a grammar defined by a set of patterns that describes the set of possible utterances (see Article IC1). In addition to knowledge about the possible structures of sentences, the natural- language-processing paradigm requires that other types of knowledge (such as a dictionary, domain knowledge, and world knowledge) be included in the programs if they are to model human thought. The most successful and promising work in natural language has aimed at developing increasingly comprehensive representations of knowledge.

The State of the Art

The present state of the art in implemented, running, natural language understanding programs is represented by: Kaplan's GSP, Wilks's (1974) work on English to French translation, Schank's MARGIE, Winograd's SHRDLU, and Wood's LUNAR (see Articles IC3c, IE2, IF3, IF5, IF4, respectively). These systems all use extensive parsing mechanisms and internal representations of the meaning extracted from sentences. They tackle a much wider range of language features than the early natural language systems, but they still deal with knowledge on a simple level. Though they take a wider view than the early systems, of what is needed to extract the meaning of a sentence, these new systems still have limited scope in terms of the language features they can "understand."

The main issues in natural language processing today are in the design of control mechanisms to flexibly integrate diverse sources of knowledge and in the development of new systems for representing and manipulating knowledge. In existent natural language systems, developed in the late 1960s, the result of processing a sentence is the addition of a unit of knowledge to a growing database. (SHRDLU manipulates a toy world but treats these actions as side-effects.) Current research stresses understanding a sentence in the context of a larger body of knowledge and building and manipulating integrated knowledge-structures. Mechanisms such as conceptualizations (Chafe, 1972), SCRIPTS (Abelson, 1973), DEMONS (Charniak, 1973), RINGS (McDermott, 1973), partitioned semantic nets (Hendrix, 1975), and KRL (Bobrow & Winograd, 1976) are attempts to develop mechanisms for integrating the meaning of sentences into larger knowledge-structures.

References

See Abelson, 1973; Bobrow & Winograd, 1977; Chafe, 1972; Charniak, 1973; Hendrix, 1976; Hewitt, 1969; Hewitt, 1971; Kaplan, 1973; Kellogg, 1968; McDermott, 1973; Quillian, 1968; Raphael, 1968; Simon, 1966; Thompson, 1968; Weaver, 1949; Wilks, 1974; Wilks, 1976; Winograd, 1972; Winograd, 1974; Winograd, 1977; Woods, 1968.

5.3.3 HARPY (Sect. VI.E.1)

Harpy Speech Recognition System

Introduction

The Harpy speech recognition program was designed at Carnegie-Mellon University by Bruce Lowerre (Lowerre, 1976) based on observations of the Hearsay I and Dragon speech systems. The representation of the search space as a network was an adaptation from the Dragon system. The Harpy network includes every syntactic and phonemic variation for all legal utterances specified by the grammar. The dynamic programming technique of the Dragon system, which utilized a complete search of the network, was modified to search only the "best few" nodes at each point in an utterance. Segmentation of the acoustical signal according to transitions in the character of the waveform rather than at fixed time intervals was a carry-over from the Hearsay I system. The combination and refinement of ideas from these two previous systems has produced the best performance results to date for a speech recognition system.

Special Features

Beam search of a "precompiled" network

All phonetic spellings for each syntactic path in the grammar are combined with word junction phenomena (the adjustments made in the pronunciations of words due to those preceding and following them in continuous speech) to create one large network. Unlike the Dragon system, which searches every path in the network, the Harpy system uses a threshold to limit the active states to only those states whose probability is within epsilon of the highest state probability for that time segment. Thus, if the probabilities are well separated, then only a few states will be considered, and, conversely, if the probabilities are bunched together (i.e., the correct selection is ambiguous), then many states will be chosen.

Processes segmented speech

The decision to use a flexible division of the acoustic signal according to acoustic events, rather than according to a fixed time interval, allows for a single acoustic template per phone. This avoids the problem of special templates for the recognition of partially completed acoustic phenomena. However, since the network is composed of a sequential phonetic representation, the system is very sensitive to missing or poorly labeled segments.

Uses heuristics to limit search time and size of network

The search time is limited by using previously computed partial search results. The network is condensed by removing redundant states, or by recognizing common groupings of states. The number of states is slightly increased, but the number of connections (i.e., pointers) can be markedly decreased, by introducing special states at common intersections in the network.

Probabilities computed during utterance

In contrast to the Dragon system, which used preselected transition probabilities for the network, the Harpy system adapts probabilities dynamically in relation to the length of the acoustic segment being processed.

Representation of Knowledge

Harpy uses only acoustic, phonetic, and syntactic knowledge sources. These sources are initially represented as a BNF grammar specification, a dictionary, and as inter-word juncture rules, which are then compiled into the finite-state transition network structure. Speaker-dependent acoustic-phonetic templates and a table of data-dependent transition probabilities are other data structures.

Limitations

The extension to much larger vocabularies must be examined in future research efforts, since the explicit creation of the network of possibilities can have a large memory requirement. The design of the current system cannot easily accommodate the inherent semantics and pragmatics of the utterance, which may be needed to constrain search in an expanded speech domain.

Summary

A case study of two dissimilar speech systems led to a first in system design: a system that approached the 1971 goals of the speech community (Licklider, et al, 1973). This was done by combining dynamic programming techniques with useful heuristics, such as beam search. The maximum capabilities of this approach and including these techniques in a larger heuristic-based system, Hearsay II, is under study at CMU. The results of the Harpy system on several test data sets are shown below. A comparison of Harpy to the 1971 goals appears in the Speech Overview article.

Harpy results: [table to be supplied]